The Consultative Committee for Space Data Systems

Recommendation for Space Data System Standards

MISSION OPERATIONS MESSAGE ABSTRACTION LAYER

RECOMMENDED STANDARD

CCSDS 521.0-B-3

BLUE BOOK

March 2024

The Consultative Committee for Space Data Systems

Recommendation for Space Data System Standards

MISSION OPERATIONS MESSAGE ABSTRACTION LAYER

RECOMMENDED STANDARD

CCSDS 521.0-B-3

BLUE BOOK

March 2024

AUTHORITY

Issue:	Recommended Standard, Issue 3
Date:	March 2024
Location:	Washington, DC, USA

This document has been approved for publication by the Management Council of the Consultative Committee for Space Data Systems (CCSDS) and represents the consensus technical agreement of the participating CCSDS Member Agencies. The procedure for review and authorization of CCSDS documents is detailed in *Organization and Processes for the Consultative Committee for Space Data Systems* (CCSDS A02.1-Y-4), and the record of Agency participation in the authorization of this document can be obtained from the CCSDS Secretariat at the email address below.

This document is published and maintained by:

CCSDS Secretariat
National Aeronautics and Space Administration
Washington, DC, USA
Email: secretariat@mailman.ccsds.org

STATEMENT OF INTENT

The Consultative Committee for Space Data Systems (CCSDS) is an organization officially established by the management of its members. The Committee meets periodically to address data systems problems that are common to all participants, and to formulate sound technical solutions to these problems. Inasmuch as participation in the CCSDS is completely voluntary, the results of Committee actions are termed **Recommended Standards** and are not considered binding on any Agency.

This **Recommended Standard** is issued by, and represents the consensus of, the CCSDS members. Endorsement of this **Recommendation** is entirely voluntary. Endorsement, however, indicates the following understandings:

- Whenever a member establishes a CCSDS-related **standard**, this **standard** will be in accord with the relevant **Recommended Standard**. Establishing such a **standard** does not preclude other provisions which a member may develop.
- Whenever a member establishes a CCSDS-related **standard**, that member will provide other CCSDS members with the following information:
  - The **standard** itself.
  - The anticipated date of initial operational capability.
  - The anticipated duration of operational service.
- Specific service arrangements shall be made via memoranda of agreement. Neither this **Recommended Standard** nor any ensuing **standard** is a substitute for a memorandum of agreement.

No later than five years from its date of issuance, this **Recommended Standard** will be reviewed by the CCSDS to determine whether it should: (1) remain in effect without change; (2) be changed to reflect the impact of new technologies, new requirements, or new directions; or (3) be retired or canceled.

In those instances when a new version of a **Recommended Standard** is issued, existing CCSDS-related member standards and implementations are not negated or deemed to be non-CCSDS compatible. It is the responsibility of each member to determine when such standards or implementations are to be modified. Each member is, however, strongly encouraged to direct planning for its new standards and implementations towards the later version of the Recommended Standard.

FOREWORD

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. CCSDS has processes for identifying patent issues and for securing from the patent holder agreement that all licensing policies are reasonable and non-discriminatory. However, CCSDS does not have a patent law staff, and CCSDS shall not be held responsible for identifying any or all such patent rights.

Through the process of normal evolution, it is expected that expansion, deletion, or modification of this document may occur. This Recommended Standard is therefore subject to CCSDS document management and change control procedures, which are defined in *Organization and Processes for the Consultative Committee for Space Data Systems* (CCSDS A02.1-Y-4). Current versions of CCSDS documents are maintained at the CCSDS Web site:

http://www.ccsds.org/

Questions relating to the contents or status of this document should be sent to the CCSDS Secretariat at the email address indicated on page i.

At time of publication, the active Member and Observer Agencies of the CCSDS were:

Member Agencies

- Agenzia Spaziale Italiana (ASI)/Italy.
- Canadian Space Agency (CSA)/Canada.
- Centre National d'Etudes Spatiales (CNES)/France.
- China National Space Administration (CNSA)/People's Republic of China.
- Deutsches Zentrum für Luft- und Raumfahrt (DLR)/Germany.
- European Space Agency (ESA)/Europe.
- Federal Space Agency (FSA)/Russian Federation.
- Instituto Nacional de Pesquisas Espaciais (INPE)/Brazil.
- Japan Aerospace Exploration Agency (JAXA)/Japan.
- National Aeronautics and Space Administration (NASA)/USA.
- UK Space Agency/United Kingdom.

Observer Agencies

- Austrian Space Agency (ASA)/Austria.
- Belgian Science Policy Office (BELSPO)/Belgium.
- Central Research Institute of Machine Building (TsNIIMash)/Russian Federation.
- China Satellite Launch and Tracking Control General, Beijing Institute of Tracking and Telecommunications Technology (CLTC/BITTT)/China.
- Chinese Academy of Sciences (CAS)/China.
- China Academy of Space Technology (CAST)/China.
- Commonwealth Scientific and Industrial Research Organization (CSIRO)/Australia.
- Danish National Space Center (DNSC)/Denmark.
- Departamento de Ciência e Tecnologia Aeroespacial (DCTA)/Brazil.
- Electronics and Telecommunications Research Institute (ETRI)/Korea.
- Egyptian Space Agency (EgSA)/Egypt.
- European Organization for the Exploitation of Meteorological Satellites (EUMETSAT)/Europe.
- European Telecommunications Satellite Organization (EUTELSAT)/Europe.
- Geo-Informatics and Space Technology Development Agency (GISTDA)/Thailand.
- Hellenic National Space Committee (HNSC)/Greece.
- Hellenic Space Agency (HSA)/Greece.
- Indian Space Research Organization (ISRO)/India.
- Institute of Space Research (IKI)/Russian Federation.
- Korea Aerospace Research Institute (KARI)/Korea.
- Ministry of Communications (MOC)/Israel.
- Mohammed Bin Rashid Space Centre (MBRSC)/United Arab Emirates.
- National Institute of Information and Communications Technology (NICT)/Japan.
- National Oceanic and Atmospheric Administration (NOAA)/USA.
- National Space Agency of the Republic of Kazakhstan (NSARK)/Kazakhstan.
- National Space Organization (NSPO)/Chinese Taipei.
- Naval Center for Space Technology (NCST)/USA.
- Netherlands Space Office (NSO)/The Netherlands.
- Research Institute for Particle & Nuclear Physics (KFKI)/Hungary.
- Scientific and Technological Research Council of Turkey (TUBITAK)/Turkey.
- South African National Space Agency (SANSA)/Republic of South Africa.
- Space and Upper Atmosphere Research Commission (SUPARCO)/Pakistan.
- Swedish Space Corporation (SSC)/Sweden.
- Swiss Space Office (SSO)/Switzerland.
- United States Geological Survey (USGS)/USA.

DOCUMENT CONTROL

Document	Title	Date	Status
CCSDS 521.0-B-1	Mission Operations Message Abstraction Layer, Recommended Standard, Issue 1	October 2010	Original issue (superseded)
CCSDS 521.0-B-2	Mission Operations Message Abstraction Layer, Recommended Standard, Issue 2	March 2013	Issue 2 (superseded)
CCSDS 521.0-B-3	Mission Operations Message Abstraction Layer, Recommended Standard, Issue 3	March 2024	Current issue: – simplifies and clarifies text as part of an overall effort to improve MO framework specifications; – corrects major deviations from CCSDS style guidelines; – attempts to better differentiate normative from informative text.

NOTE – Changes from the original issue are too numerous to permit meaningful application of change bars.

CONTENTS

CONTENTS (continued)

CONTENTS (continued)

Figure Page

Table

CONTENTS (continued)

1 INTRODUCTION

1.1 GENERAL

This Recommended Standard defines the Mission Operations (MO) Message Abstraction Layer (MAL) in conformance with the service framework specified in reference [D1], Mission Operations Services Concept.

The MO MAL is a framework that provides generic service patterns to the Mission Operation services defined in reference [D1]. These MO services are defined in terms of the MAL.

1.2 PURPOSE AND SCOPE

This Recommended Standard defines, in an abstract manner, the MAL in terms of:

a) the concepts that it builds upon;

b) the basic types it provides;

c) the Message Headers required by the layer;

d) the relationship between, and the valid sequence of, the messages and resulting behaviours.

It does not specify:

a) individual implementations or products;

b) the implementation of entities or interfaces within real systems;

c) the methods or technologies required for communications.

1.3 DOCUMENT STRUCTURE

This Recommended Standard is organized as follows:

a) section 1 provides purpose and scope and lists definitions, conventions, and references used throughout the Recommended Standard;

b) section 2 presents an overview of the concepts;

c) section 3 specifies the Interaction Patterns used to define services;

d) section 4 is a formal specification of the MAL data structures;

e) section 5 is a formal specification of the MO Errors;

f) section 6 is the formal service specification Extensible Markup Language (XML) schema;

g) section 7 presents the information required to be specified in a concrete deployment;

h) section 8 specifies the formal compliance requirements of the MAL.

1.4 DEFINITION OF TERMS

software component (component): A software unit containing the business function. Components offer their function as services, which can either be used internally or which can be made available for use outside the component through Provided Service Interfaces. Components may also depend on services provided by other components through Consumed Service Interfaces.

service: A set of capabilities that a component provides to another component via an interface. A service is defined in terms of the set of operations that can be invoked and performed through the Service Interface. Service specifications define the capabilities, behaviour, and external interfaces, but do not define the implementation.

Service Interface: A set of interactions provided by a component for participation with another component for some purpose, along with constraints on how they can occur. A Service Interface is an external interface of a service in which the behaviour of the Service Provider component is exposed. Each service will have one defined 'Provided Service Interface', and may have one or more 'Consumed Service Interface' and one 'Management Service Interface'.

Provided Service Interface: A Service Interface that exposes the service function contained in a component for use by Service Consumers. It receives the MAL messages from a Consumed Service Interface and maps them into API calls on the provider component.

Consumed Service Interface: The API presented to the consumer component that maps from the service operations to one or more service data units(s) contained in MAL messages that are transported to the Provided Service Interface.

Management Service Interface: A Service Interface that exposes management functions of a service function contained in a component for use by Service Consumers.

Area: The functional context of a group of related services. It is used as a taxonomy means for grouping a set of related services and data. Examples of Areas are Monitoring and Control, Mission Planning, Telerobotics, Navigation, Automation.

domain: A means of applying hierarchical taxonomy to mission operation related data. The term domain is used with different meanings in specific mission and deployment scenarios in order to be able to distinguish between data from different sources. For instance, in a constellation, the domain may be used to distinguish the data from individual spacecraft of the constellation. In a different context the domain may be used to distinguish between different applications, subsystems, and payloads.

MO Object: Representation of a complex data type with two specific characteristics: MO Objects have a unique and immutable identity and can be referenced unambiguously. MO Objects are used to represent and formally specify the complex data model in service specifications. Subsection 4.4 provides a detailed definition and explanation of the MO Object and its characteristics.

Service Provider (provider): A component that offers a service to another by means of one of its Provided Service Interfaces.

Service Consumer (consumer): A component that consumes or uses a service provided by another component. A component may be a provider of some services and a consumer of others.

service data unit, SDU: A unit of data that is sent by a Service Interface, and is transmitted semantically unchanged, to a peer Service Interface.

binding: The access mechanism for a service. Bindings are used to locate and access Service Interfaces. Services use bindings to describe the access mechanisms that consumers have to use to call the service. The binding specifies unambiguously the protocol stack required to access a Service Interface. Bindings may be defined statically at compile time or they may use a variety of dynamic run-time mechanisms (DNS, ports, discovery).

Service Capability Set: A grouping of the service operations that remains sensible and coherent, and also provides a Service Provider with an ability to communicate to a consumer its capability. The specification of services is based on the expectation that different deployments require different levels of complexity and functionality from a service. To this end, a given service may be implementable at one of several distinct levels, corresponding to the inclusion of one or more capability sets.

Service Connection (connection): A communications connection between a Consumed Service Interface and a Provided Service Interface over a specific Binding. When a component consumes the services of a provider component, this link is known as a Service Connection (or a connection).

Service Extension: Addition of capabilities to a base service. A service may extend the capabilities of another service with additional operations. An extended service is indistinguishable from the base service to consumers such that consumers of the base service can also be consumers of the extended service without modification.

broker: In a Publish-Subscribe Interaction Pattern, an intermediary component that decouples the publishers of messages from the subscribers.

1.5 NOMENCLATURE

1.5.1 NORMATIVE TEXT

The following conventions apply for the normative specifications in this Recommended Standard:

a) the words ‘shall’ and ‘must’ imply a binding and verifiable specification;

b) the word ‘should’ implies an optional, but desirable, specification;

c) the word ‘may’ implies an optional specification;

d) the words ‘is’, ‘are’, and ‘will’ imply statements of fact.

NOTE – These conventions do not imply constraints on diction in text that is clearly informative in nature.

1.5.2 INFORMATIVE TEXT

In the normative sections of this document, informative text is set off from the normative specifications either in notes or under one of the following subsection headings:

– Overview;

– Background;

– Rationale;

– Discussion.

1.6 REFERENCES

The following publications contain provisions which, through reference in this text, constitute provisions of this document. At the time of publication, the editions indicated were valid. All publications are subject to revision, and users of this document are encouraged to investigate the possibility of applying the most recent editions of the publications indicated below. The CCSDS Secretariat maintains a register of currently valid CCSDS publications.

[1] *Mission Operations Reference Model*. Recommendation for Space Data System Standards, CCSDS 520.1-M-1. Magenta Book. Issue 1. Washington, D.C.: CCSDS, July 2010.

[2] "MIME Media Types." Internet Assigned Numbers Authority. http://www.iana.org/assignments/media-types.

[3] *IEEE Standard for Floating-Point Arithmetic*. 3rd ed. IEEE Std 754-2019. New York: IEEE, 2019.

[4] Julie D. Allen, et al., eds. *The Unicode Standard*. Version 15.0 – Core Specification. Mountain View, California: The Unicode Consortium, September 2022.

[5] T. Berners-Lee, R. Fielding, and L. Masinter. *Uniform Resource Identifier (URI): Generic Syntax*. STD 66. Reston, Virginia: ISOC, January 2005.

[6] H. Thompson and C. Lilley. *XML Media Types*. RFC 7303. Reston, Virginia: ISOC, July 2014.

[7] "Registries." Space Assigned Numbers Authority. https://sanaregistry.org/r.

[8] *Mission Operations—Common Services*. Issue 1. Recommendation for Space Data System Standards (Blue Book), CCSDS 522.0-B-1. Washington, D.C.: CCSDS, May 2020.

[9] *Security Guide for Mission Planners*. Issue 2. Report Concerning Space Data System Standards (Green Book), CCSDS 350.7-G-2. Washington, D.C.: CCSDS, April 2019.

NOTE – Informative references are listed in annex D.

2 OVERVIEW

2.1 GENERAL

The MO specifications detail a layered service-oriented architecture (see figure 2-1). This architecture is described in the Mission Operation Reference Model book (reference [1]). This book specifies the Message Abstraction Layer part of this architecture.

MO services form a contract between a consumer of a service and the provider of that service. Each operation of a service has a set behaviour: a message is sent from the consumer to the provider to instigate the operation, and zero or more messages can be exchanged thereafter, depending on the specific pattern and operation. This set of messages, and the pattern in which they are exchanged, needs to be defined for each operation of each service.

An Interaction Pattern details a standard exchange pattern that removes the need for each operation to detail its exchange pattern individually. By defining a pattern and stating that a given operation is defined in terms of that pattern, the operation definition can focus on the specifics of that operation and rely on the standard pattern to facilitate the interaction.

The MAL defines this set of standard Interaction Patterns as an abstract interface that is used by the MO services.

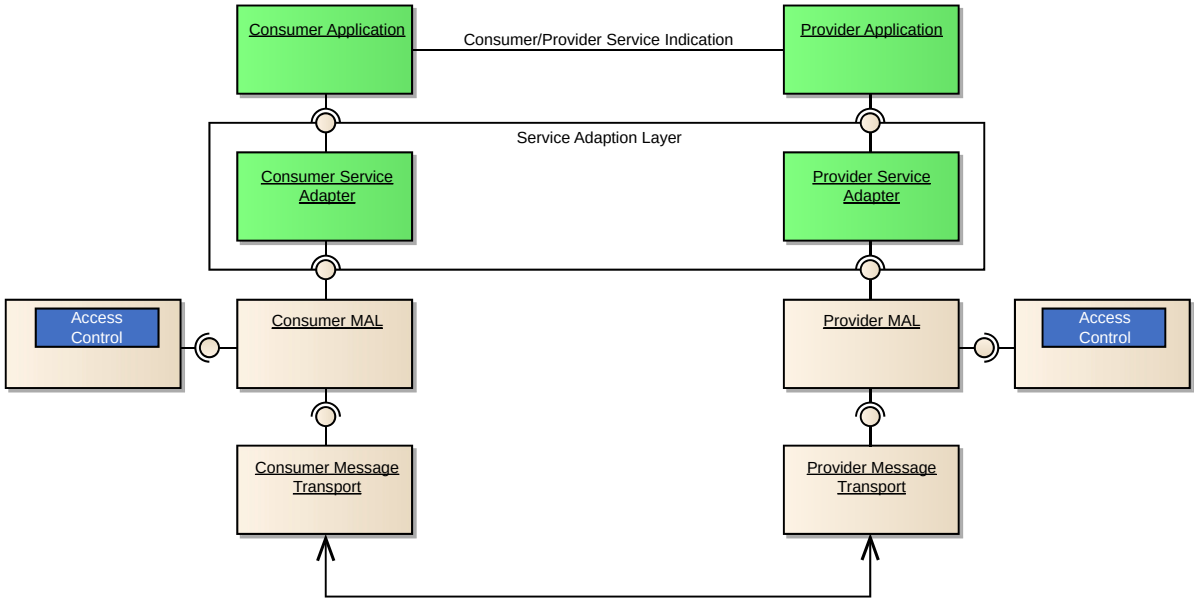

Figure 2-1: MO Layered Architecture

The MAL has three interfaces to the neighbouring components of the Architecture: the Service Adaption Layer above it, the Transport Layer below it, and the Access Control. This specification standardizes two of these interfaces:

– the interface of the MAL that is presented to the service's layer;

– the interface between the MAL and the Access Control.

The interface to the Transport Layer is deliberately not standardized. Although this interface could provide portability of software components, it would not provide end-to-end interoperability between Agencies. The interoperability of the exchanged messages at Transport Layer is achieved by the compliance of the MAL to a concrete Transport Binding Blue Book.

Section 7 details the requirements for achieving end-to-end interoperability in a concrete deployment of the MO layered architecture.

The MO service specifications are defined using the MAL. They do not contain any specific information on the programming language or the transport/encoding to use. However, in a concrete deployment (e.g., a ground system for a space mission scenario), moving from the abstract to concrete, it is required to define a Language Mapping that states how the abstract MAL and MO service specifications are to be realized in some specific language: this defines the API in that language. In addition, a transport mapping is required that defines the mapping from the abstract MAL data structures into a specific and unambiguous encoding of the messages and to a defined and unambiguous mapping to a specific data transport. It is only when these mappings are defined that is possible to implement services that use the MAL interface and use the transport bindings to exchange data. (For further information on this, see reference [1].)

This document contains the formal specification for these abstract interfaces and the abstract data types that they reference. More information about the MO Concept can be found in reference [D1], and more information about the Reference Model can be found in reference [1].

2.2 ABSTRACT INTERFACE SPECIFICATIONS

2.2.1 GENERAL

Each abstract interface is defined as a set of Interaction Patterns. For each pattern defined, there is a common layout of the document section.

The following subsections describe the sections and diagram formats specific to the Interaction Patterns.

2.2.2 PATTERN OVERVIEW

The overview section of the pattern contains a message exchange sequence, which shows the interaction sequence between a source consumer and a destination provider.

The main aspects to note are the direction of the arrows (indicating message direction) and the order of the messages (top down). The bars under the consumer and provider show synchronicity of the message exchange.

In the following example (figure 2-2), it can be seen that the consumer sends an initial message to the provider, which responds with an acknowledgement. The acknowledgment is in response to the initial message, as shown by the connected bars under the consumer and provider.

At some time in the future, the provider will send another response:

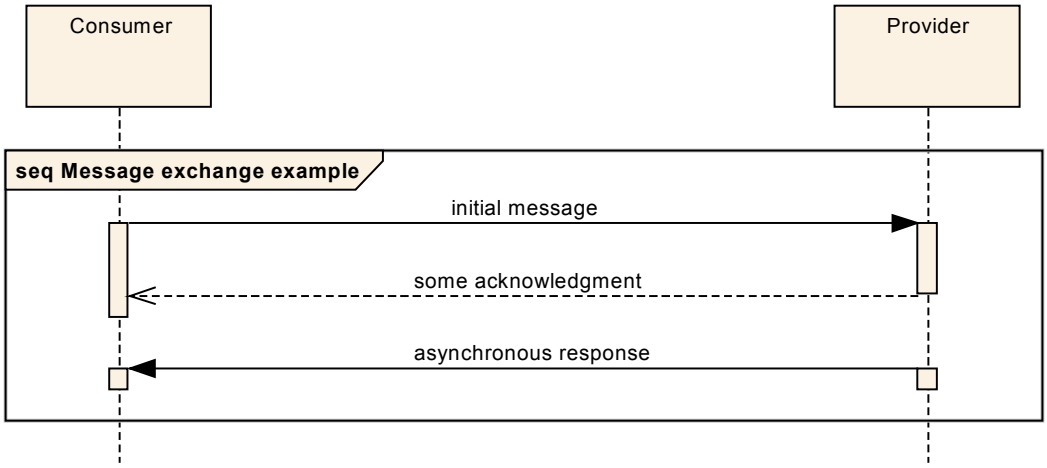

Figure 2-2: Message Exchange Sequence Example

Because the pattern shows the response, the consumer can expect it, but because the bars are not connected, the timing is considered to be indeterminate. This means that although the message will arrive, it cannot be determined when arrival will occur. It is important to note that any synchronicity shown is concerned only with messages; it does not imply, or require, a synchronous (blocking) behaviour in either the client or provider, as these are implementation issues.

2.2.3 DESCRIPTION AND USAGE

The Description and Usage sections are used to describe the pattern and define the expected usage, respectively.

2.2.4 ERROR HANDLING

The Error Handling section defines the positions in the pattern where errors are permitted to be generated. It uses the same sequence diagram notation as the nominal pattern sequence from the Overview section.

2.2.5 OPERATION TEMPLATE

Each Interaction Pattern definition contains a table that defines the template for operations that use that pattern:

Table 2-1: Example Operation Template

Operation Identifier	<<Operation name>>		
Interaction Pattern	<<Interaction pattern name>>		
Pattern Sequence	Message	Nullable	Type Signature
<<Message direction>>	<<Message name>>	<<Nullability>>	<<Message Signature*>>
…	…		…

The message direction denotes the direction of the message relative to the provider of the pattern and is either IN or OUT. So all messages directed towards the provider are IN messages, and all messages directed away from the provider are OUT messages. It is expected that message names match those defined in the Primitives section.

Blue cells (dark grey when printed on a monochrome printer) contain table headings, light grey cells contain fields that are fixed for a pattern, and white cells contain values that must be provided by the operation or structure.

The Type Signature column contains the list of types that make up the Body of a particular message and the respective field names. Zero to many types may be listed here (indicated by the asterisk in table 2-1) and together define the body of the indicated message.

This document defines a data type specification language that is specified as MAL data types. The Type Signatures are defined using the MAL data types to define the various base data types, the notation used to describe them, and the rules for their combination into complex data structures. This is separate from the encoding technology used which is responsible for mapping the abstract data type notation from the data type specification into an actual ‘bits and bytes’ representation.

Even though the Body Types in this specification are defined using the MAL data types, any data type specification language (such as XML schema) may be used in an actual service specification which requires a separate step of mapping the MAL data types to the specific Programming Language data types and of mapping the MAL Interaction Patterns to a concrete data transport protocol.

2.2.6 PRIMITIVES

Each pattern defines a set of primitives in a table as below:

Table 2-2: Example Primitive List

Primitive
<<Primitive name>>

For each listed primitive, there exists a **Request**, **Message**, and **Indication** of the same name. The Requests and Indications are used by the following state diagrams. Requests are transitions that are triggered by the component being modelled, Messages are the entities that are transferred between the two components to progress the interaction, and Indications are transitions that are triggered by an external activity (see figure 2-3).

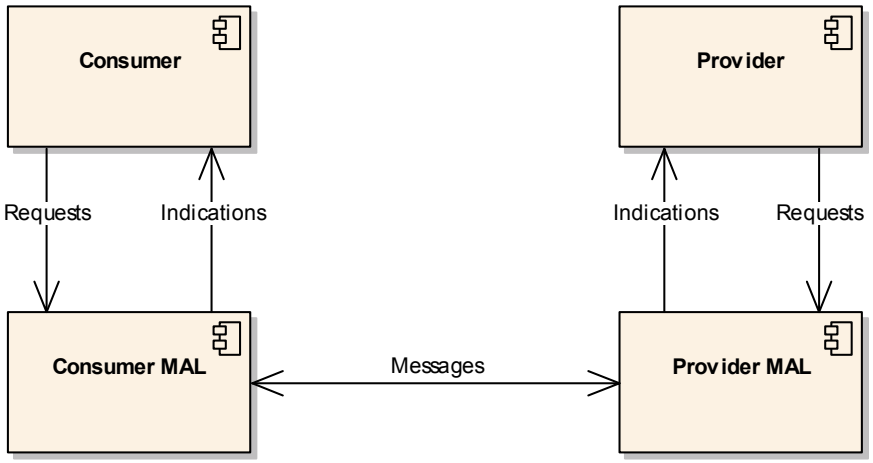

Figure 2-3: Request, Indication, and Message Relationship

Therefore a component issues Requests, which cause Messages to be transmitted, which raise Indications in the destination component.

2.2.7 STATE CHARTS

For each defined interaction, there can also be specified one or more state diagrams (see figure 2-4). The diagrams define the allowed states and transitions between those states.

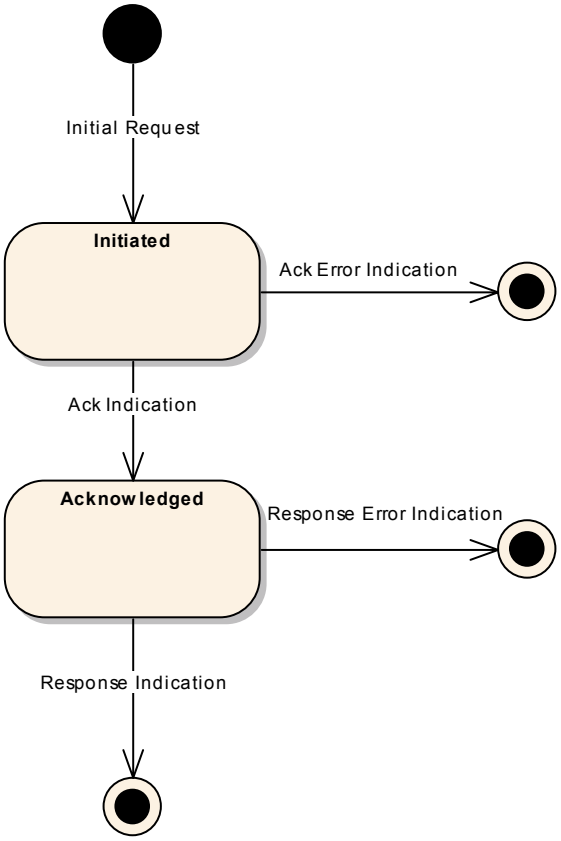

Figure 2-4: Consumer State Diagram Example

The states are those of the MAL of the relevant component and apply to a single instance of a pattern; for example, figure 2-4 shows the allowed states and transitions for a specific pattern on the consumer, but it is the responsibility of the consumer MAL to maintain that state information and reject invalid state transitions.

It should be noted that, in a concrete MAL implementation in a given deployment, there can be many interactions concurrently occurring. In that case, the MAL implementation would need to be able to track the state of each interaction instance independently.

Figure 2-4 shows the state diagram for the consumer component from the example in figure 2-2 and includes some error state transitions that are not shown in the example.

2.2.8 REQUESTS AND INDICATIONS

2.2.8.1 General

For each pattern, there is a set of Requests that can be issued and a set of Indications that can be received. These sets are defined in the Primitives section for the pattern. The allowed state transitions are defined in the State Charts section for the pattern.

Each pattern defines the following subsections for each primitive.

2.2.8.2 Function

The Function subsection defines the use of a specific Request or Indication.

2.2.8.3 Semantics

The Semantics subsection defines what information is required for the Request or Indication. It is equivalent to the arguments of a programming language function call.

2.2.8.4 When Generated

The When Generated subsection specifies the circumstances that trigger the generation of the Request or Indication.

2.2.8.5 Effect on Reception

The Effect on Reception subsection specifies the effect(s) of the reception of a Request by a MAL or of an Indication by an Application. For a Request, it is usually the transmission of a Message to the destination; for an Indication, it is pattern specific.

2.2.8.6 Message Header

The MAL defines a standard Message Header that is used to manage the interaction. The Message Header subsection specifies the values of the fields of the Message Header to be used for the message. The subsection can specify all the fields, or only fields that are different from another Request/Indication.

The full definition of the Message Header structure is provided in 3.5.

2.2.9 MESSAGE EXAMPLE DECOMPOSITIONS

Each pattern is illustrated with an example which shows how each message will look when decomposed into a packet-like structure. It is important to note that the implemented version of such a type of structure is completely dependent on the chosen encoding and transport bindings adopted for each deployment.

Each message example is split into two distinct parts, the Message Header and the Message Body, as presented in figure 2-5. The Message Body may have two parts, a Standard Pattern Body, and/or a Service-Specific Body.

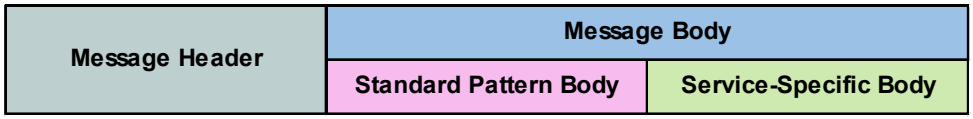

Figure 2-5: Message Decomposition Key

NOTE – The Standard Pattern Body is only used in the messages of the Publish-Subscribe Interaction Pattern.

Each message part is then broken down into its composing fields. For example, the Message Header is decomposed into a set of fields as presented in figure 2-6:

Message Header												
From	Authentication Id	To	Timestamp	Interaction Type	Interaction Stage	Transaction Id	Service Area	Service	Operation	Area Version	Is Error Message	Supplements
Provider	SC X	Broker		PUBSUB	3	123	Example	Example	testPubSub	1	FALSE	

Figure 2-6: Message Header Decomposition Example

Field values are provided as an example, and as such, more human-readable values are used, rather than numeric equivalents that may be used in highly efficient encodings. The detailed definitions of the Message Header fields shown in this example are specified in 3.5.

NOTE – The example for the Timestamp field is omitted from the decomposition examples. In an actual exchange, this field would be stamped with an exact time.

Figure 2-7 shows an example of three fields in the Service-Specific Body parts of a message. Fields might have complex structures within, such as field3 with a TestNotify structure that includes two nested fields (name and value) as presented below:

field1	field2	field3	
Boolean	**Integer**	**TestNotify**	
		name	value
		Identifier	**Integer**
FALSE	33	text	1234

Figure 2-7: Message Body Decomposition Example

NOTE – The structures used in the body of the examples for each pattern are only examples, and as such, no explanation of contained values or meanings is provided or required.

2.3 REPRESENTING SERVICE SPECIFICATIONS

Each service specification includes an overview table that details the Area and Service identifiers and then lists the operations of that service (see table 2-3).

Table 2-3: Service Overview Table

Area Identifier	Service Identifier	Area Number	Service Number	Area Version
Interaction Pattern	Operation Identifier		Operation Number	Capability Set

The values of table 2-3 are used to populate the Service Area, Service, Service Operation, and Area Version fields of the Message Header. The Area Number, Service Number, and Operation Number fields provide an alternative numerical identification scheme parallel to the identifier scheme.

The choice of whether to use identifiers, numbers, or another alternative in a concrete transport is transport-specific, as long as it is possible to correctly determine the area, service, operation, and version values from the transmitted information. The numbers are expected to be used by transports that require a more efficient identification mechanism than identifiers and will not overlap with existing service definitions. The set of permissible values is in the UShort range and starts at ‘1’.

The Area Version is a number that is used to differentiate between issues of an area. Initially, it will be set to the number ‘1’, and future updates to the service will increase that number. The set of permissible values is in the UOctet range.

The Capability Set field is a numerical identifier that holds the Service Capability Set number for that operation. Operations that hold the same Capability Set number within a service specification are considered to be part of the same Service Capability Set. The set of permissible values is in the UShort range.

3 ABSTRACT SERVICE SPECIFICATIONS

3.1 OVERVIEW

There are two abstract interfaces defined in this specification. The first is the abstract interface of the MAL. The MAL abstract interface defines the interactions that are presented to the higher layers, the information and interaction between the two interacting components, and also the expected behaviour of the Application Layer that uses the MAL. It is detailed in 3.6.

The second is the abstract interface of the Access Control component. To support the Access Control and security aspects of the reference model in reference [1], the MAL requires a standard abstract interface to an Access Control component. The Access Control component only has the single interaction that is used by the MAL as defined in the Reference Model (reference [1]). The implementation of the Access Control component is deployment-specific, as is the security policy and rules to be used, which should be chosen according to reference [9]; however, the interaction with that component is part of the MAL Standard. The concept of the generic Access Control Interface and the delegation of responsibilities are detailed in 3.7.

General constraints that apply to all patterns are defined in 3.2, 3.4, and 3.5.

3.2 DOMAINS

3.2.1 BACKGROUND

A domain is a hierarchical taxonomy to mission operation related data. The term domain is used with different meanings in specific mission and deployment scenarios in order to be able to distinguish data from different sources. Hence, the domain is expected to remain static for the data coming from a deployed provider. For instance, in a constellation of satellites, the domain may be used to distinguish the data from each individual satellite of the constellation. Moreover, in a different context, the domain may be used to distinguish data between different applications, subsystems, and payloads.

3.2.2 REQUIREMENTS

3.2.2.1 A Domain shall be defined as a list of identifiers.

3.2.2.2 The most significant domain part shall be listed first in the list (e.g., Agency), and each subsequent domain identifier in the list narrows the preceding domain.

3.2.2.3 The domain shall not have any of its individual domain identifier parts set as the empty Identifier ‘’ nor the full stop character ‘.’.

NOTE – The full stop character ‘.’ is typically used as a separator between the individual domain parts when presenting it to a user and sometimes this representation is also used in configuration files.

3.2.2.4 The domain shall not have any of its individual domain identifier parts set as the asterisk character '*'.

NOTE – The asterisk character '*' is allowed to be used on domain filters for operations, as it can represent a wildcard to a particular individual domain identifier (e.g., in consumer subscriptions to the broker). This enables consumers to fine-tune the domains that they intend to subscribe or filter on.

3.2.3 DISCUSSION

This subsection describes a few examples of how the domain is envisaged to be used in concrete scenarios. These are just examples for clarification purposes, and the implementers are free to choose the domain as long as it respects the requirements in 3.2.2. The examples will use the full stop character '.' in order to separate the individual domain parts.

Example 1—Two domains on a ground-to-ground interoperability scenario between two agencies:

- agencyA.missionX
- agencyB.missionY

Example 2—Three domains on a mission with a constellation of three satellites:

- agencyA.missionX.satellite1
- agencyA.missionX.satellite2
- agencyA.missionX.satellite3

Example 3—Two domains on a mission with one satellite with an onboard computer and a dedicated payload computer:

- agencyA.missionX.obc
- agencyA.missionX.payloadcomputer

When defining the domain, one should try to maintain a sane and logical breakdown that makes sense for that particular scenario. Domain hierarchies should be kept to a sensible minimum as presented in the examples above.

3.3 TRANSACTION HANDLING

3.3.1 The Message Header shall contain a 'Transaction Id' field providing a mechanism for the originating MAL of a message exchange to uniquely identify the response, or set of responses, to a message.

3.3.2 The 'Transaction Id' shall be unique for each instance of an Interaction Pattern, except in the PUBLISH-SUBSCRIBE Interaction Pattern.

3.3.3 The 'Transaction Id' shall be:

a) used by the originating MAL to identify messages in a particular instance of an Interaction Pattern; and

b) returned by the service provider in matching messages (returns from submits, etc.).

3.3.4 The originating MAL shall ensure that the combination of the following Message Header fields form a unique index: 'Transaction Id', 'From', 'Area', 'Area version', 'Service', and 'Operation'.

3.3.5 The PUBLISH-SUBSCRIBE Interaction Pattern shall have different Transaction Ids for the same Interaction Pattern instance as defined in 3.6.6.

3.4 STATE TRANSITIONS

3.4.1 Each pattern shall specify the set of states that both the provider and consumer can attain during the execution of the pattern.

3.4.2 To move between the states, each pattern shall define the set of legal transitions either as a Request or an Indication.

3.4.3 For a Request to be issued, the source entity shall be in the correct state.

3.4.4 Issuing a Request when in the incorrect state shall cause an INCORRECT_STATE error to be raised by the local MAL, and the pattern shall end at this point.

3.4.5 For Indications, the receiving entity shall be required to be in the correct state.

3.4.6 Reception of an Indication in a state other than the correct one shall cause an INCORRECT_STATE error to be raised by the local MAL, and the pattern shall end at this point.

3.5 MAL MESSAGE COMPOSITION

3.5.1 GENERAL

Each MAL message shall be composed of two parts: the Message Header and the Message Body. Figure 3-1 is a visual representation of a MAL message. The Message Body may have two parts, a Standard Pattern Body and/or a Service-Specific Body.

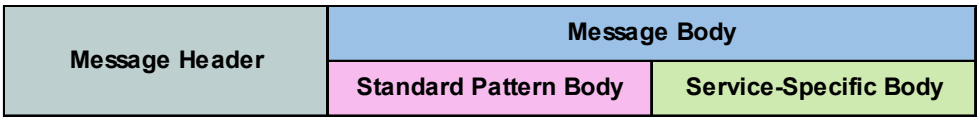

Figure 3-1: MAL Message Composition

NOTE – The Standard Pattern Body is only used in the messages of the Publish-Subscribe Interaction Pattern.

3.5.2 MESSAGE HEADER

3.5.2.1 The Message Header shall have the fields specified in table 3-1, and these fields shall be used in all Interaction Patterns.

3.5.2.2 Each Interaction Pattern shall set the values of the fields in table 3-1, with the relevant values defined in the Request and Indication sections for that pattern as defined in 3.6. This includes the fields 'Interaction Type', 'Interaction Stage', and 'Is Error Message'.

Table 3-1: MAL Message Header Fields

Field	Type	Description
From	Identifier	Message Source
Authentication Id	Blob	Authentication Identifier of Message Originator
To	Identifier	Message Destination
Timestamp	Time	Message generation timestamp
Interaction Type	InteractionType	Interaction Pattern Type
Interaction Stage	UOctet	Interaction Pattern Stage
Transaction Id	Long	Unique to consumer
Service Area	UShort	Service Area
Service	UShort	Service
Operation	UShort	Service Operation
Area version	UOctet	Area version
Is Error Message	Boolean	'True' if this is an error message; else 'False'
Supplements	List<NamedValue>	Optional supplementary fields

3.5.2.3 The 'From' and 'To' fields are identifiers. MAL makes no assumption about what type of addressing information is used to indicate the source and destination of MAL messages. For a particular implementation/deployment, these fields may contain a URI-formatted text (reference [5]), a textual name (such as the name of an Agency), or a number (e.g., indicating an Application Identifier [APID]). How these values are used for routing of the messages from source to destination is deployment and implementation specific and must be specified in an out-of-band agreement such as an interface control document.

3.5.2.4 The 'Authentication Id' field shall hold the authentication identifier of the originator of the message. The use of this field is implementation- and deployment-specific. Examples of its usage are provided in 3.7.

3.5.2.5 The 'Timestamp' field shall be set as the time when the MAL Message is created.

3.5.2.6 The 'Transaction Id' field shall be set according to 3.2.

3.5.2.7 The 'Service Area', 'Service', 'Operation', and 'Area Version' fields shall be set according to the defined service specification and respective operation in use.

3.5.2.8 The 'Supplements' field may contain a set of optional named values in the form of a list of names and respective values. The MAL does not define any default set of supplement names. If, for a particular deployment, supplement fields are to be used, these shall be captured in an out-of-band agreement.

3.5.2.9 A 'Supplements' field with no named value shall be represented by an empty NamedValue list.

NOTE – The set of named values contained in the 'Supplements' field of MAL messages may vary from one message to another. Even when out-of-band agreements fix minimal constraints on this set, the MAL shall accept messages with additional named values in this field.

3.5.3 MESSAGE BODY

3.5.3.1 The Message Body shall be encoded according to the encoding specification selected for a certain deployment.

3.5.3.2 The Message Body shall be structured in accordance with the Nullable column and the Type Signature column of the operation template in a service specification.

3.5.3.3 Zero to many types may be listed in the Type Signature column, each type being one part of the Message Body; together they form the complete body of the indicated message.

3.5.3.4 The listed types should use the MAL data type specification as defined in section 4 of this specification, or the service-specific types derived from these.

3.5.4 MO ERROR MESSAGE BODY

3.5.4.1 A service specification shall define which MO Errors may be returned for a specific operation and if any extra information is provided with the error message as presented in section 5.

3.5.4.2 The service specification shall specify the data type of the extra information for the specific error.

3.5.4.3 The Message Body of an MO Error shall be composed of an Error Number followed by Extra Information as presented in figure 3-2.

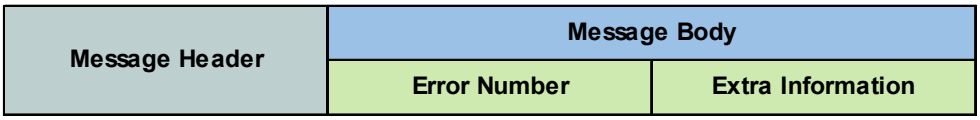

Figure 3-2: MO Error Message Body

3.5.4.4 The Error Number shall be in the UInteger values range.

3.5.4.5 The Extra Information value shall immediately follow the Error Number.

3.5.4.6 If no Extra Information is provided by an error, then the Extra Information value shall be set to NULL.

3.6 MAL SERVICE INTERFACE

3.6.1 SEND INTERACTION PATTERN

3.6.1.1 Overview

3.6.1.1.1 General

The SEND pattern is the most basic interaction (figure 3-3); it is a single message from the consumer to the provider. No acknowledgement is sent by the provider.

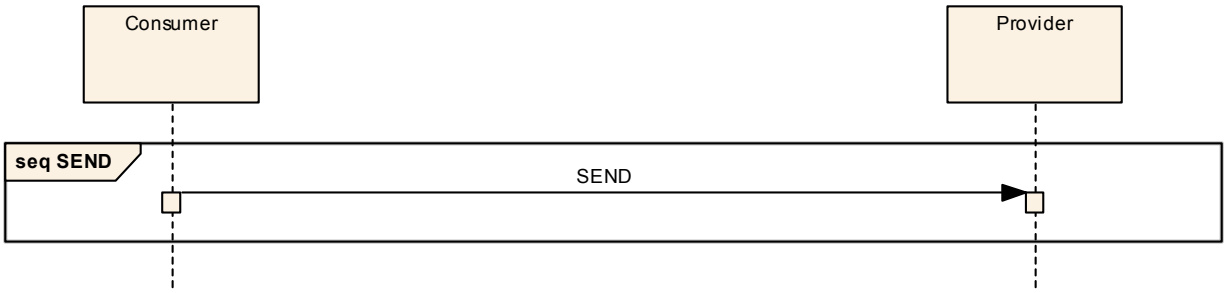

Figure 3-3: SEND Interaction Pattern Message Sequence

3.6.1.1.2 Description

The SEND pattern is the basic interaction of which all other patterns can be considered extensions. It is the simple passing of a message from a consumer to a provider. No return message is sent from the provider to the consumer, so the consumer has no indication the provider has received the message.

3.6.1.2 Usage

The SEND pattern should be used for non-critical messages when the possible loss of one or more of these messages is not considered critical. An example would be regular redundant status messages.

3.6.1.3 Error Handling

MO Errors (see section 4) may be generated by the consumer service layers, but no error may be generated by the provider, as the Interaction Pattern does not allow for the provider to return an error.

3.6.1.4 Operation Template

The SEND pattern shall conform to the SEND operation template defined in table 3-2.

Table 3-2: SEND Operation Template

Operation Identifier	<<Operation name>>		
Interaction Pattern	SEND		
Pattern Sequence	Message	Nullable	Type Signature
IN	SEND	<<Nullability>>	<< Send Signature>>

NOTE – The SEND operation template only contains the operation name and the type signature containing both the type of the structure and respective field name that is sent as the Message Body.

3.6.1.5 Primitives

The SEND pattern shall use the primitive defined in table 3-3.

Table 3-3: SEND Primitive List

Primitive
SEND

3.6.1.6 State Charts

3.6.1.6.1 Consumer Side

The state chart contains only one transition from the initial state to the final one, so it is not represented. This transition shall be triggered by the primitive SEND Request.

3.6.1.6.2 Provider Side

The state chart contains only one transition from the initial state to the final one, so it is not represented. This transition shall be triggered by the primitive SEND Indication.

3.6.1.7 SEND Requests and Indications

3.6.1.7.1 Function

3.6.1.7.1.1 The SEND Request primitive shall be used by the consumer application to initiate a SEND interaction.

3.6.1.7.1.2 The SEND Indication primitive shall be used by the provider MAL to deliver a SEND Message to a provider and initiate a SEND interaction.

3.6.1.7.2 Semantics

SEND Request and SEND Indication shall provide parameters as follows:

(SEND Message Header, SEND Message Body)

3.6.1.7.3 When Generated

3.6.1.7.3.1 A SEND Request may be generated by the consumer application at any time.

3.6.1.7.3.2 A SEND Indication shall be generated by the provider MAL upon reception of a SEND Message, once checked via the Access Control interface, from a consumer.

3.6.1.7.4 Effect on Reception

3.6.1.7.4.1 Reception of a SEND Request shall, once checked via the Access Control interface, cause the consumer MAL to initiate a SEND interaction by transmitting a SEND Message to the provider.

3.6.1.7.4.2 The pattern shall end at this point for the consumer.

3.6.1.7.4.3 On reception of a SEND Indication a provider shall process the operation.

3.6.1.7.4.4 The pattern shall end at this point for the provider.

3.6.1.7.5 Message Header

3.6.1.7.5.1 For the SEND Message, the Message Header fields shall be as defined in 3.5 except for the fields in table 3-4.

3.6.1.7.5.2 The contents of the SEND Message body, as specified in the operation template, shall immediately follow the Message Header.

Table 3-4: SEND Message Header Fields

Field	Value
From	Consumer identifier
Authentication Id	Consumer Authentication Identifier
To	Provider identifier
Interaction Type	SEND
Interaction Stage	1
Transaction Id	Provided by consumer MAL

3.6.1.8 Discussion

The following example shows a simple service with a single SEND pattern-based operation:

Operation Identifier	testSend		
Interaction Pattern	SEND		
Pattern Sequence	Message	Nullable	Type Signature
IN	SEND	Yes	TestBody body

The TestBody structure is defined as follows:

Name	TestBody		
Extends	Composite		
Short Form Part	Example Only		
Field	Type	Nullable	Comment
FirstItem	String	Yes	Example String item
SecondItem	Integer	Yes	Example Integer item

NOTE – The structure presented above follows the representation of Composite structures specified in 4.2.4.

This corresponds to the following message:

Message Header													TestBody	
From	Authentication Id	To	Timestamp	Interaction Type	Interaction Stage	Transaction Id	Service Area	Service	Operation	Area Version	Is Error Message	Supplements	FirstItem	SecondItem
Consumer	Op A	Provider		SEND	1	123	Example	Example	testSend	1	FALSE		Hello	1

NOTE – An actual service specification, rather than the example given here, would fully specify the meaning of, and required values of, all structures used by the service.

3.6.2 SUBMIT INTERACTION PATTERN

3.6.2.1 Overview

3.6.2.1.1 General

The SUBMIT pattern is a simple confirmed message exchange pattern (figure 3-4). The consumer sends a message to a provider and the provider acknowledges it.

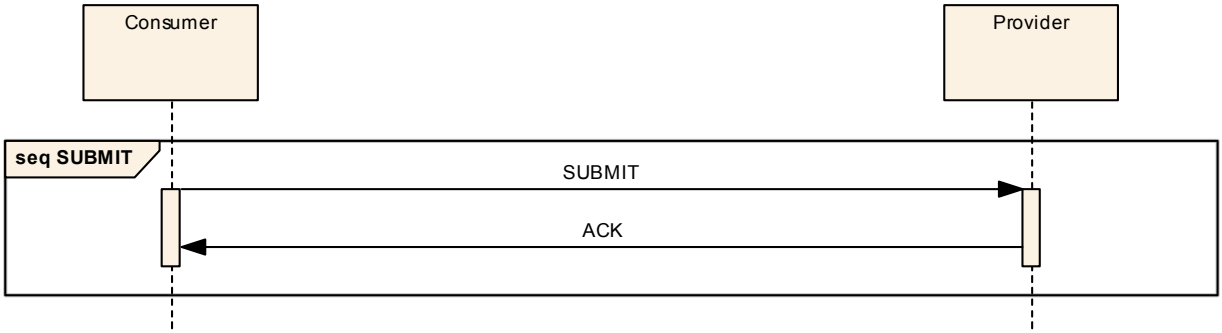

Figure 3-4: SUBMIT Interaction Pattern Message Sequence

The meaning of the return acknowledgement is operation specific. Each operation specification details the exact meaning of the operation's return acknowledgement for its context.

3.6.2.1.2 Description

The SUBMIT pattern extends the SEND pattern by providing a return acknowledgment message from the provider back to the consumer. The service specification details the meaning of the acknowledgment message for a specific operation.

3.6.2.2 Usage

The SUBMIT pattern should be used for simple operations that complete quickly but must be confirmed to the consumer.

3.6.2.3 Error Handling

3.6.2.3.1 The return acknowledgment shall be replaced with an error message (see section 4) if an error occurs during the processing of the operation as shown in figure 3-5.

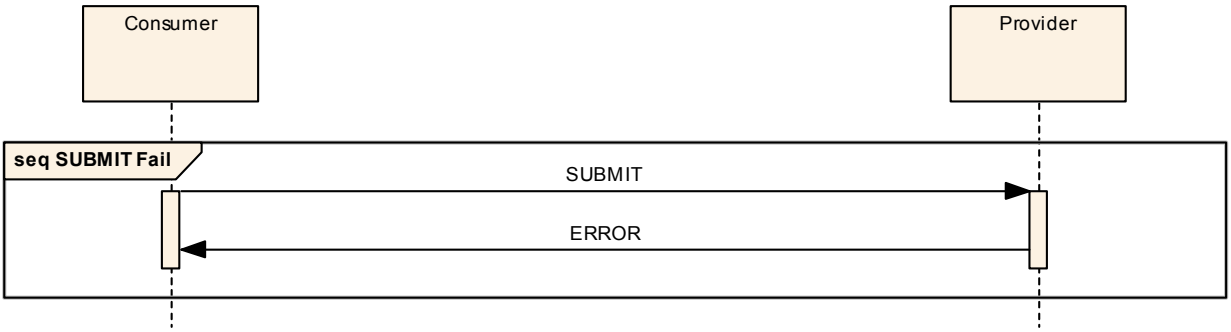

Figure 3-5: SUBMIT Interaction Pattern Error Sequence

3.6.2.3.2 Either the acknowledgment or the error message shall be returned, but never both.

3.6.2.3.3 The service specification shall specify that the acknowledgement is not returned until all processing that can generate an error has been completed, if a service is required to be able to return an error during the processing of the message.

3.6.2.4 Operation Template

3.6.2.4.1 The SUBMIT pattern shall conform to the SUBMIT operation template defined in table 3-5.

Table 3-5: SUBMIT Operation Template

Operation Identifier	<<Operation name>>		
Interaction Pattern	SUBMIT		
Pattern Sequence	Message	Nullable	Type Signature
IN	SUBMIT	<<Nullability>>	<< Submit Signature>>

NOTES

1 The SUBMIT template only contains the operation name and the type signature containing both the type of the structure and respective field name that is submitted as the SUBMIT Message body.

2 To keep the operation as simple as possible, the return acknowledgement message does not have a body. Because of this, it is not shown in the operation template.

3.6.2.4.2 If a service-defined return message is required, for example, to return an identifier for the operation, then the REQUEST pattern should be used instead of the SUBMIT pattern.

3.6.2.5 Primitives

The SUBMIT pattern shall use the primitives defined in table 3-6.

Table 3-6: SUBMIT Primitive List

Primitive
SUBMIT
ACK
ERROR

3.6.2.6 State Charts

3.6.2.6.1 Consumer Side

3.6.2.6.1.1 Figure 3-6 is the consumer side state chart for the SUBMIT pattern.

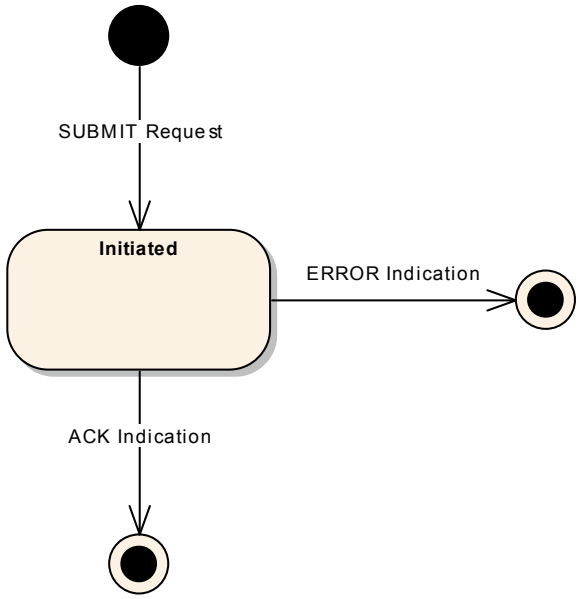

Figure 3-6: SUBMIT Consumer State Chart

3.6.2.6.1.2 The initial transition shall be triggered by the primitive SUBMIT Request and shall lead to the Initiated State.

3.6.2.6.1.3 There are two possible transitions from the Initiated State:

a) the first is the indication of an acknowledgement, ACK Indication, and shall lead to the final state;

b) the second is the indication of an error, ERROR Indication, and shall lead to the final state.

3.6.2.6.2 Provider Side

3.6.2.6.2.1 Figure 3-7 is the provider side state chart for the SUBMIT pattern.

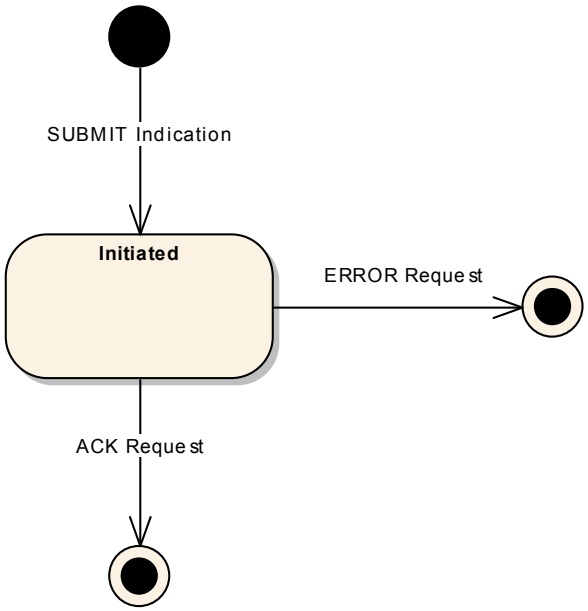

Figure 3-7: SUBMIT Provider State Chart

3.6.2.6.2.2 The initial transition shall be triggered by the primitive SUBMIT Indication and shall lead to the Initiated State.

3.6.2.6.2.3 There are two possible transitions from the Initiated State:

a) the first is the transmission of an acknowledgement, ACK Request, and shall lead to the final state;

b) the second is the transmission of an error, ERROR Request, and shall lead to the final state.

3.6.2.7 Requests and Indications

3.6.2.7.1 SUBMIT

3.6.2.7.1.1 Function

3.6.2.7.1.1.1 The SUBMIT Request primitive shall be used by the consumer application to initiate a SUBMIT interaction.

3.6.2.7.1.1.2 The SUBMIT Indication primitive shall be used by the provider MAL to deliver a SUBMIT Message to a provider and initiate a SUBMIT interaction.

3.6.2.7.1.2 Semantics

SUBMIT Request and SUBMIT Indication shall provide parameters as follows:

(SUBMIT Message Header, SUBMIT Message Body)

3.6.2.7.1.3 When Generated

3.6.2.7.1.3.1 A SUBMIT Request may be generated by the consumer application at any time.

3.6.2.7.1.3.2 A SUBMIT Indication shall be generated by the provider MAL upon reception of a SUBMIT Message, once checked via the Access Control interface, from a consumer.

3.6.2.7.1.4 Effect on Reception

3.6.2.7.1.4.1 Reception of a SUBMIT Request shall, once checked via the Access Control interface, cause the consumer MAL to initiate a SUBMIT interaction by transmitting a SUBMIT Message to the provider.

3.6.2.7.1.4.2 The consumer MAL shall then enter the Initiated State.

3.6.2.7.1.4.3 On reception of a SUBMIT Indication by the provider, the provider MAL shall enter the Initiated State.

3.6.2.7.1.4.4 At this point, the provider shall start processing the operation.

3.6.2.7.1.5 Message Header

3.6.2.7.1.5.1 For the SUBMIT Message, the Message Header fields shall be as defined in 3.5 except for the fields in table 3-7.

3.6.2.7.1.5.2 The contents of the Message Body, as specified in the operation template, shall immediately follow the Message Header.

Table 3-7: SUBMIT Message Header Fields

Field	Value
From	Consumer identifier
Authentication Id	Consumer Authentication Identifier
To	Provider identifier
Interaction Type	SUBMIT
Interaction Stage	1
Transaction Id	Provided by consumer MAL

3.6.2.7.2 ACK

3.6.2.7.2.1 Function

3.6.2.7.2.1.1 The ACK Request primitive shall be used by the provider application to acknowledge a SUBMIT interaction.

3.6.2.7.2.1.2 The ACK Indication primitive shall be used by the consumer MAL to deliver an ACK Message to a consumer.

3.6.2.7.2.2 Semantics

ACK Request and ACK Indication shall provide parameters as follows:

(ACK Message Header)

3.6.2.7.2.3 When Generated

3.6.2.7.2.3.1 An ACK Request shall be used by the provider application with the provider MAL in the Initiated State to acknowledge a SUBMIT interaction.

3.6.2.7.2.3.2 An ACK Indication shall be generated by the consumer MAL in the Initiated State upon reception of an ACK Message, once checked via the Access Control interface, from a provider.

3.6.2.7.2.4 Effect on Reception

3.6.2.7.2.4.1 Reception of an ACK Request shall, once checked via the Access Control interface, cause the provider MAL to transmit an ACK Message to the consumer.

3.6.2.7.2.4.2 The pattern shall end at this point for the provider.

3.6.2.7.2.4.3 On reception of an ACK Indication, a consumer shall end the Interaction Pattern with success.

3.6.2.7.2.5 Message Header

For the ACK Message, the Message Header fields shall be as defined in 3.5 except for the fields in table 3-8.

Table 3-8: Submit ACK Message Header Fields

Field	Value
From	Provider identifier
Authentication Id	Provider Authentication Identifier
To	Consumer identifier
Interaction Type	SUBMIT
Interaction Stage	2
Transaction Id	Transaction Id from initial message

3.6.2.7.3 ERROR

3.6.2.7.3.1 Function

3.6.2.7.3.1.1 The ERROR Request primitive shall be used by the provider application to end a SUBMIT interaction with an error.

3.6.2.7.3.1.2 The ERROR Indication primitive shall be used by the consumer MAL to deliver an ERROR Message to a consumer.

3.6.2.7.3.2 Semantics

ERROR Request and ERROR Indication shall provide parameters as follows:

(ERROR Message Header, Error Number, Extra Information)

3.6.2.7.3.3 When Generated

3.6.2.7.3.3.1 An ERROR Request shall be used by the provider application with the provider MAL in the Initiated State to transmit an error.

3.6.2.7.3.3.2 An ERROR Indication shall be generated by the consumer MAL in the Initiated State upon one of two events:

a) the reception of an ERROR Message, once checked via the Access Control interface, from a provider;

b) an error raised by the local communication layer.

3.6.2.7.3.4 Effect on Reception

3.6.2.7.3.4.1 Reception of an ERROR Request shall, once checked via the Access Control interface, cause the provider MAL to transmit an ERROR Message to the consumer.

3.6.2.7.3.4.2 The pattern shall end at this point for the provider.

3.6.2.7.3.4.3 On reception of an ERROR Indication, a consumer shall end the interaction with an error.

3.6.2.7.3.5 Message Header

3.6.2.7.3.5.1 For the ERROR Message, the Message Header fields shall be the same as for the ACK Message except for the Is Error Message field, which is set to TRUE.

3.6.2.7.3.5.2 The Error Number and any extra information (referred to as the 'Error Information' from this point onwards) shall immediately follow the Message Header.

3.6.2.8 Discussion

The following shows a simple example service with a single SUBMIT pattern-based operation:

Operation Identifier	testSubmit		
Interaction Pattern	SUBMIT		
Pattern Sequence	Message	Nullable	Type Signature
IN	SUBMIT	Yes	TestBody body

The TestBody structure is defined in 3.6.1.8. This corresponds to the following message being transmitted:

Message Header													TestBody	
From	Authentication Id	To	Timestamp	Interaction Type	Interaction Stage	Transaction Id	Service Area	Service	Operation	Area Version	Is Error Message	Supplements	FirstItem	SecondItem
Consumer	Op A	Provider		SUBMIT	1	123	Example	Example	testSubmit	1	FALSE		Hello	1

And it corresponds to the following acknowledgement being returned:

Message Header												
From	Authentication Id	To	Timestamp	Interaction Type	Interaction Stage	Transaction Id	Service Area	Service	Operation	Area Version	Is Error Message	Supplements
Provider	SC X	Consumer		SUBMIT	2	123	Example	Example	testSubmit	1	FALSE	

NOTE – An actual service specification, rather than the example given here, would fully specify the meaning of, and required values of, all structures used by the service.

3.6.3 REQUEST INTERACTION PATTERN

3.6.3.1 Overview

3.6.3.1.1 General

The REQUEST pattern extends the SUBMIT pattern by replacing the simple acknowledgement with a data response message (figure 3-8). No acknowledgement other than the data response is sent.

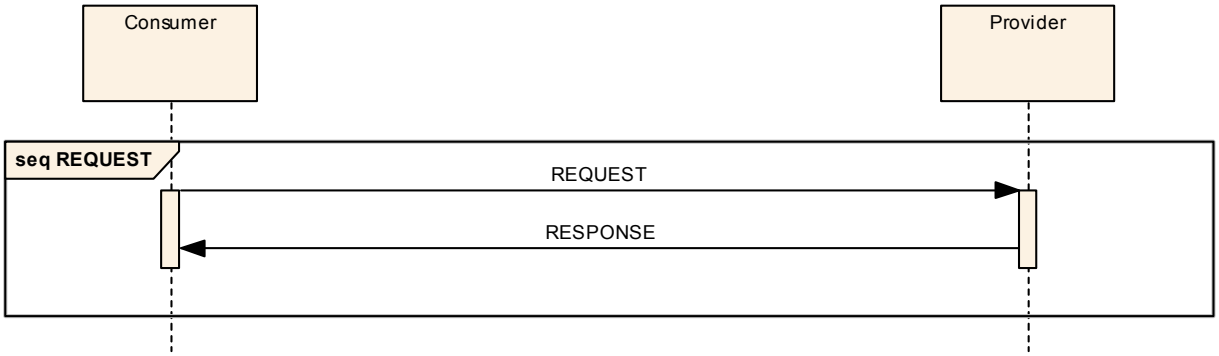

Figure 3-8: REQUEST Interaction Pattern Message Sequence

3.6.3.1.2 Description

The REQUEST pattern provides a simple request/response message exchange. Unlike the SUBMIT pattern, no acknowledgement is sent upon reception of the request; however, a data response is sent. The lack of an acknowledgement with only the return data response for this pattern means that it is primarily expected to be used for situations in which the operation takes minimal time.

3.6.3.2 Usage

The REQUEST pattern should be used only for operations that are completed in a relatively short period of time. If a more extended or indeterminate period is possible, then the more advanced INVOKE or PROGRESS patterns should be specified.

3.6.3.3 Error Handling

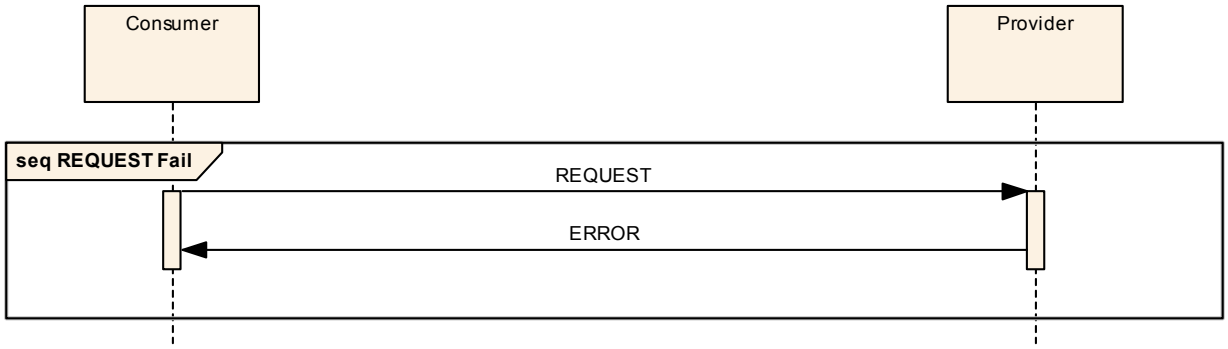

Figure 3-9: REQUEST Interaction Pattern Error Sequence

3.6.3.3.1 The data response shall be replaced with an error message (see section 4) if an error occurs during the processing of the operation (figure 3-9).

3.6.3.3.2 Either the data response or the error message shall be returned, but never both.

3.6.3.4 Operation Template

The REQUEST pattern shall conform to the REQUEST operation template defined in table 3-9.

Table 3-9: REQUEST Operation Template

Operation Identifier	<<Operation name>>		
Interaction Pattern	REQUEST		
Pattern Sequence	Message	Nullable	Type Signature
IN	REQUEST	<<Nullability>>	<<Request Signature>>
OUT	RESPONSE	<<Nullability>>	<<Response Signature>>

NOTE – The REQUEST pattern template extends the SUBMIT template by adding the requirement for a return message signature.

3.6.3.5 Primitives

The REQUEST pattern shall use the primitives defined in table 3-10.

Table 3-10: REQUEST Primitive List

Primitive
REQUEST
RESPONSE
ERROR

3.6.3.6 State Charts

3.6.3.6.1 Consumer Side

3.6.3.6.1.1 Figure 3-10 is the consumer side state chart for the REQUEST pattern.

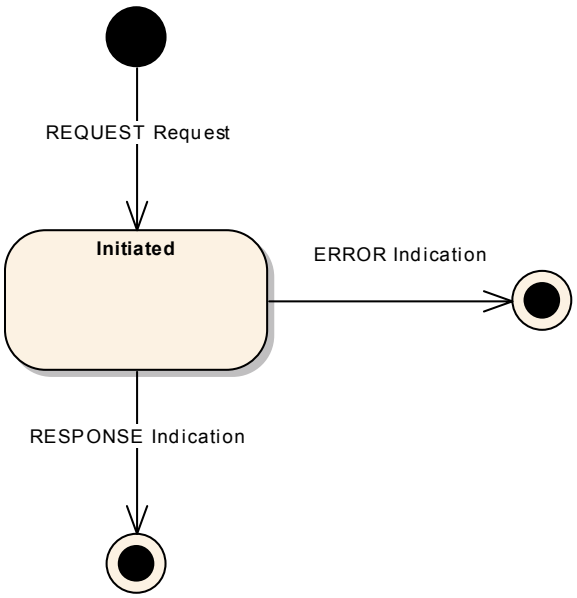

Figure 3-10: REQUEST Consumer State Chart

3.6.3.6.1.2 The initial transition shall be triggered by the primitive REQUEST Request and shall lead to the Initiated State.

3.6.3.6.1.3 There are two possible transitions from the Initiated State:

a) the first is the indication of a response, RESPONSE Indication, and shall lead to the final state;

b) the second is the indication of an error, ERROR Indication, and shall lead to the final state.

3.6.3.6.2 Provider Side

3.6.3.6.2.1 Figure 3-11 is the provider side state chart for the REQUEST pattern.

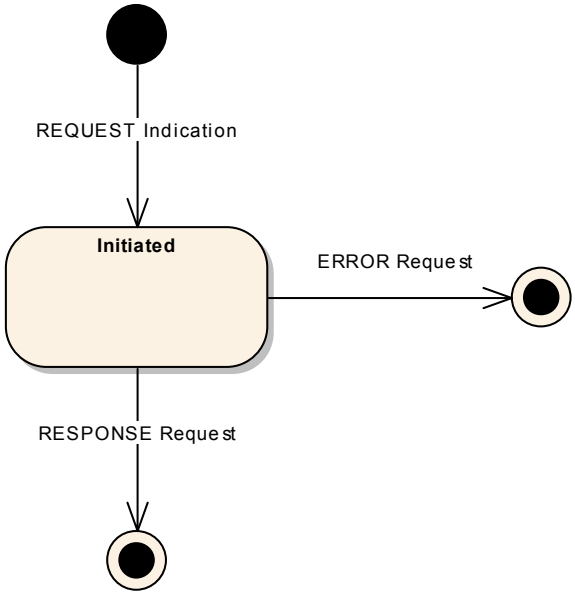

Figure 3-11: REQUEST Provider State Chart

3.6.3.6.2.2 The initial transition shall be triggered by the primitive REQUEST Indication and shall lead to the Initiated State.

3.6.3.6.2.3 There are two possible transitions from the Initiated State:

a) the first is the transmission of a response, RESPONSE Request, and shall lead to the final state;

b) the second is the transmission of an error, ERROR Request, and shall lead to the final state.

3.6.3.7 Requests and Indications

3.6.3.7.1 REQUEST

3.6.3.7.1.1 Function

3.6.3.7.1.1.1 The REQUEST Request primitive shall be used by the consumer application to initiate a REQUEST interaction.

3.6.3.7.1.1.2 The REQUEST Indication primitive shall be used by the provider MAL to deliver a REQUEST Message to a provider and initiate a REQUEST interaction.

3.6.3.7.1.2 Semantics

REQUEST Request and REQUEST Indication shall provide parameters as follows:

(REQUEST Message Header, REQUEST Message Body)

3.6.3.7.1.3 When Generated

3.6.3.7.1.3.1 A REQUEST Request may be generated by the consumer application at any time.

3.6.3.7.1.3.2 A REQUEST Indication shall be generated by the provider MAL upon reception of a REQUEST Message, once checked via the Access Control interface, from a consumer.

3.6.3.7.1.4 Effect on Reception

3.6.3.7.1.4.1 Reception of a REQUEST Request shall, once checked via the Access Control interface, cause the consumer MAL to initiate a REQUEST interaction by transmitting a REQUEST Message to the provider.

3.6.3.7.1.4.2 The consumer MAL shall enter the Initiated State at this point.

3.6.3.7.1.4.3 On reception of a REQUEST Indication by the provider, the provider MAL shall enter the Initiated State.

3.6.3.7.1.4.4 At this point, the provider shall start processing the operation.

3.6.3.7.1.5 Message Header

3.6.3.7.1.5.1 For the REQUEST Message, the Message Header fields shall be as defined in 3.5 except for the fields in table 3-11.

3.6.3.7.1.5.2 The contents of the Message Body, as specified in the operation template, shall immediately follow the Message Header.

Table 3-11: REQUEST Message Header Fields

Field	Value
From	Consumer identifier
Authentication Id	Consumer Authentication Identifier
To	Provider identifier
Interaction Type	REQUEST
Interaction Stage	1
Transaction Id	Provided by consumer MAL

3.6.3.7.2 RESPONSE

3.6.3.7.2.1 Function

3.6.3.7.2.1.1 The RESPONSE Request primitive shall be used by the provider application to transmit a response to a REQUEST interaction.

3.6.3.7.2.1.2 The RESPONSE Indication primitive shall be used by the consumer MAL to deliver a RESPONSE Message to a consumer.

3.6.3.7.2.2 Semantics

RESPONSE Request and RESPONSE Indication shall provide parameters as follows:

(RESPONSE Message Header, RESPONSE Message Body)

3.6.3.7.2.3 When Generated

3.6.3.7.2.3.1 A RESPONSE Request shall be used by the provider application with the provider MAL in the Initiated State to transmit a final response to a REQUEST interaction.

3.6.3.7.2.3.2 A RESPONSE Indication shall be generated by the consumer MAL in the Initiated State upon reception of a RESPONSE Message, once checked via the Access Control interface, from a provider.

3.6.3.7.2.4 Effect on Reception

3.6.3.7.2.4.1 Reception of a RESPONSE Request shall, once checked via the Access Control interface, cause the provider MAL to transmit the supplied response as a RESPONSE Message to the consumer.

3.6.3.7.2.4.2 The pattern shall end at this point for the provider.

3.6.3.7.2.4.3 On reception of a RESPONSE Indication, a consumer shall end the Interaction Pattern with success.

3.6.3.7.2.5 Message Header

3.6.3.7.2.5.1 For the RESPONSE Message, the Message Header fields shall be as defined in 3.5 except for the fields in table 3-12.

3.6.3.7.2.5.2 The contents of the Message Body, as specified in the operation template, shall immediately follow the Message Header.

Table 3-12: Request RESPONSE Message Header Fields

Field	Value
From	Provider identifier
Authentication Id	Provider Authentication Identifier
To	Consumer identifier
Interaction Type	REQUEST
Interaction Stage	2
Transaction Id	Transaction Id from initial message

3.6.3.7.3 ERROR

3.6.3.7.3.1 Function

3.6.3.7.3.1.1 The ERROR Request primitive shall be used by the provider application to end a REQUEST interaction with an error.

3.6.3.7.3.1.2 The ERROR Indication primitive shall be used by the consumer MAL to deliver an ERROR Message to a consumer.

3.6.3.7.3.2 Semantics

ERROR Request and ERROR Indication shall provide parameters as follows:

(ERROR Message Header, Error Number, Extra Information)

3.6.3.7.3.3 When Generated

3.6.3.7.3.3.1 An ERROR Request shall be used by the provider application with the provider MAL in the Initiated State to transmit an error.

3.6.3.7.3.3.2 An ERROR Indication shall be generated by the consumer MAL in the Initiated State upon one of two events:

a) the reception of an ERROR Message, once checked via the Access Control interface, from a provider;

b) an error raised by the local communication layer.

3.6.3.7.3.4 Effect on Reception

3.6.3.7.3.4.1 Reception of an ERROR Request shall, once checked via the Access Control interface, cause the provider MAL to transmit an ERROR Message to the consumer.

3.6.3.7.3.4.2 The pattern shall end at this point for the provider.

3.6.3.7.3.4.3 On reception of an ERROR Indication, a consumer shall end the interaction with an error.

3.6.3.7.3.5 Message Header

3.6.3.7.3.5.1 For the ERROR Message, the Message Header fields shall be the same as the RESPONSE Message except for the Is Error Message field, which is set to TRUE.

3.6.3.7.3.5.2 The Error Information shall immediately follow the Message Header.

3.6.3.8 Discussion

The following example shows a simple service that defines a single REQUEST operation:

Operation Identifier	testRequest		
Interaction Pattern	REQUEST		
Pattern Sequence	Message	Nullable	Type Signature
IN	REQUEST	Yes	TestBody body
OUT	RESPONSE	Yes	TestResponse response

The TestBody structure is defined in 3.6.1.8, and the TestResponse structure is defined below:

Name	TestResponse		
Extends	Composite		
Short Form Part	Example Only		
Field	Type	Nullable	Comment
RspnItem	Boolean	Yes	Example Boolean item
RspnField	Float	Yes	Example Float item

This corresponds to the following message being transmitted:

Message Header													TestBody	
From	Authentication Id	To	Timestamp	Interaction Type	Interaction Stage	Transaction Id	Service Area	Service	Operation	Area Version	Is Error Message	Supplements	FirstItem	SecondItem
Consumer	Op A	Provider		REQUEST	1	123	Example	Example	testRequest	1	FALSE		Hello	1

And it also corresponds to the following response being returned:

Message Header													TestResponse	
From	Authentication Id	To	Timestamp	Interaction Type	Interaction Stage	Transaction Id	Service Area	Service	Operation	Area Version	Is Error Message	Supplements	RspnItem	RspnField
Provider	SC X	Consumer		REQUEST	2	123	Example	Example	testRequest	1	FALSE		TRUE	31.0

NOTE – An actual service specification, rather than the example given here, would fully specify the meaning of, and required values of, all structures used by the service.

3.6.4 INVOKE INTERACTION PATTERN

3.6.4.1 Overview

3.6.4.1.1 General

The INVOKE pattern extends the REQUEST pattern to add a mandatory acknowledgement of the initial message (figure 3-12). This allows the operation to confirm the receipt of the Request before proceeding to process the Request.

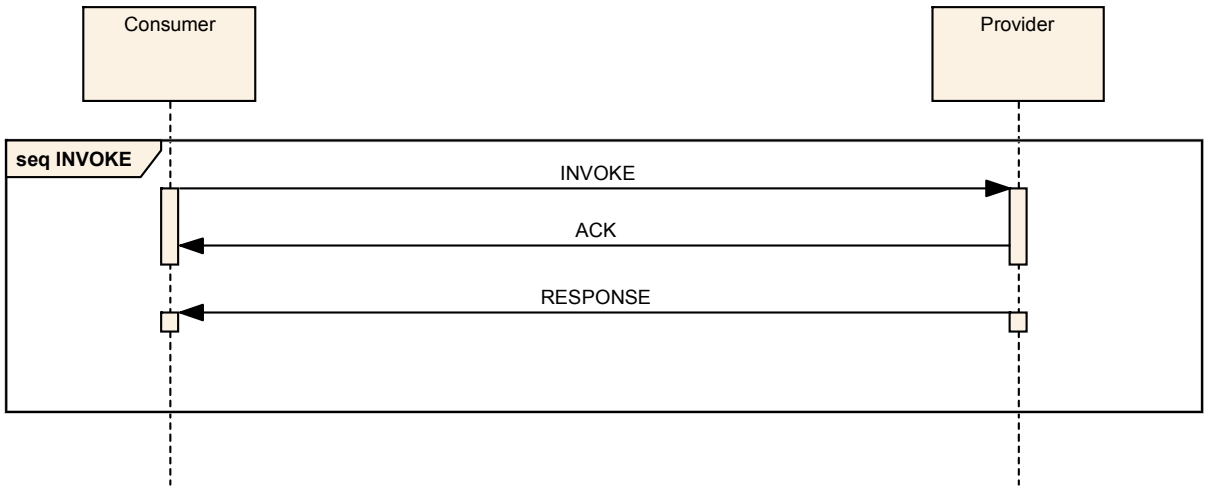

Figure 3-12: INVOKE Interaction Pattern Message Sequence

The acknowledgement message is an extension of the acknowledgment used in the SUBMIT pattern. This pattern allows the acknowledgment to return a service-specific Message Body. A service-defined acknowledgement is required because the INVOKE pattern is indeterminate. It allows an operation to provide an indication of the operation status upon INVOKE reception. An example would be validation of the INVOKE arguments.

3.6.4.1.2 Description

The INVOKE pattern extends the REQUEST pattern with the addition of a mandatory acknowledgement of the initial message.

3.6.4.2 Usage

The INVOKE pattern should be used when there is a significant or indeterminate amount of time taken to process the Request and produce the data response.

NOTE – The provision of the service-defined acknowledgement message allows an operation to return supplementary, status, or summary information about the Request before processing continues (for example, an identifier used for querying INVOKE status using another operation).

3.6.4.3 Error Handling

The INVOKE pattern may report failure in two distinct ways:

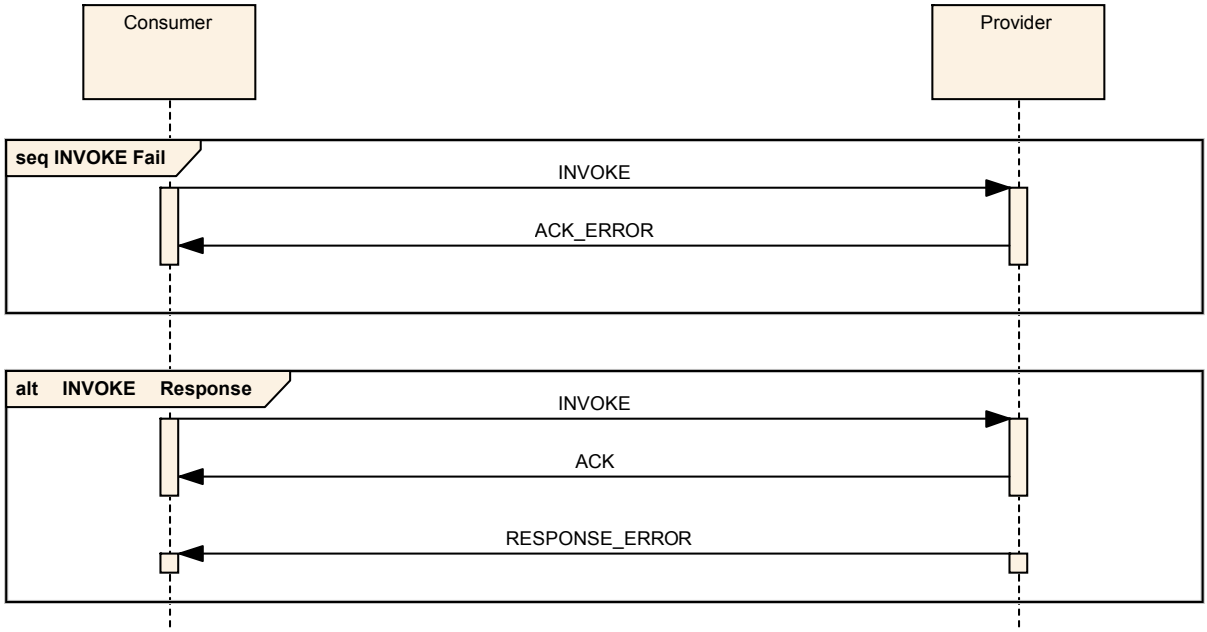

Figure 3-13: INVOKE Interaction Pattern Error Sequence

a) The provider may return an error message (see section 4) in replacement of the ACK Message, shown in the first sequence in figure 3-13.

b) The provider may return an error after the acknowledgement is sent in replacement of the RESPONSE Message, shown in the second sequence in figure 3-13.

c) After an error message is sent, no further messages shall be generated as part of the pattern.

NOTE – It is expected that the first case would be used to report an error with the INVOKE Request and the second to report an error that occurs during processing.

3.6.4.4 Operation Template

The INVOKE pattern shall conform to the INVOKE operation template defined in table 3-13

Table 3-13: INVOKE Operation Template

Operation Identifier	<<Operation name>>		
Interaction Pattern	INVOKE		
Pattern Sequence	Message	Nullable	Type Signature
IN	INVOKE	<<Nullability>>	<<Invoke Signature>>
OUT	ACK	<<Nullability>>	<<Ack Signature>>
OUT	RESPONSE	<<Nullability>>	<<Response Signature>>

NOTE – The INVOKE template extends the REQUEST pattern template by requiring an additional (synchronous) acknowledgment message between the initial message and the (asynchronous) response message.

3.6.4.5 Primitives

The SUBMIT pattern shall use the primitives defined in table 3-14.

Table 3-14: INVOKE Primitive List

Primitive
INVOKE
ACK
RESPONSE
ACK_ERROR
RESPONSE_ERROR

3.6.4.6 State Charts

3.6.4.6.1 Consumer Side

3.6.4.6.1.1 Figure 3-14 is the consumer side state chart for the INVOKE pattern.

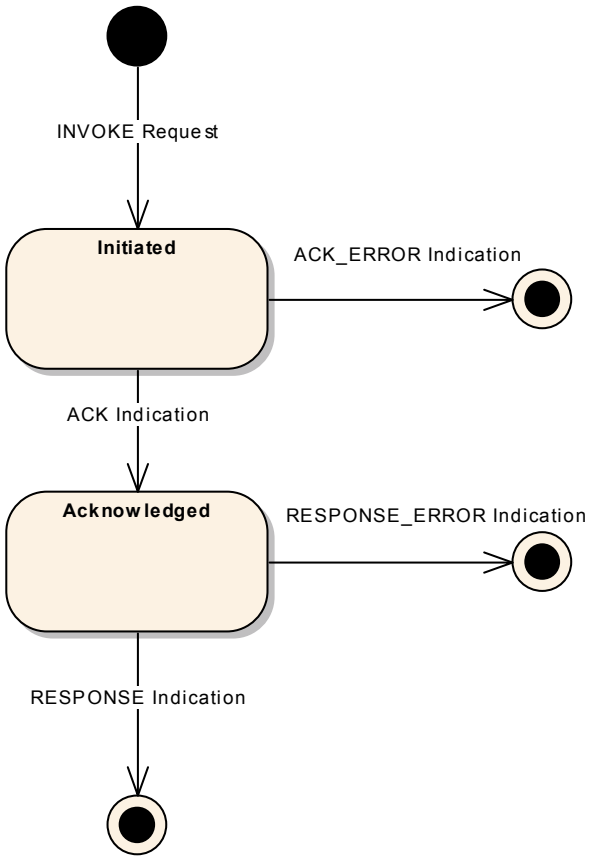

Figure 3-14: INVOKE Consumer State Chart

3.6.4.6.1.2 The initial transition shall be triggered by the primitive INVOKE Request and shall lead to the Initiated State.

3.6.4.6.1.3 There are two possible transitions from the Initiated State:

a) the first is the indication of an acknowledgement, ACK Indication, and shall lead to the Acknowledged State;

b) the second is the indication of an error, ACK_ERROR Indication, and shall lead to the final state.

3.6.4.6.1.4 There are two possible transitions from the Acknowledged State:

a) the first is the indication of a response, RESPONSE Indication, and shall lead to the final state;

b) the second is the indication of an error, RESPONSE_ERROR Indication, and shall lead to the final state.

3.6.4.6.2 Provider Side

3.6.4.6.2.1 Figure 3-15 is the provider side state chart for the INVOKE pattern.

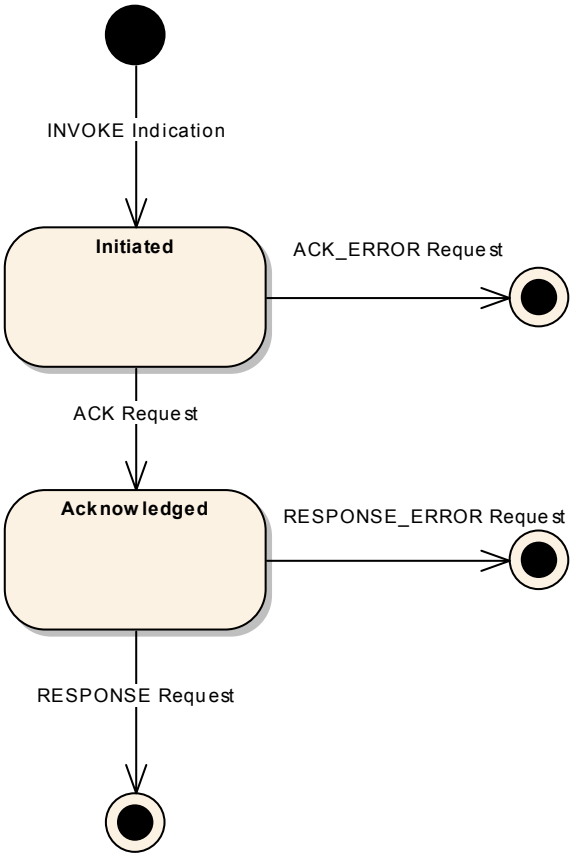

Figure 3-15: INVOKE Provider State Chart

3.6.4.6.2.2 The initial transition shall be triggered by the primitive INVOKE Indication and shall lead to the Initiated State.

3.6.4.6.2.3 There are two possible transitions from the Initiated State:

a) the first is the transmission of an acknowledgement, ACK Request, and shall lead to the Acknowledged State;

b) the second is the transmission of an error, ACK_ERROR Request, and shall lead to the final state.

3.6.4.6.2.4 There are two possible transitions from the Acknowledged State:

a) the first is the transmission of a response, RESPONSE Request, and shall lead to the final state;

b) the second is the transmission of an error, RESPONSE_ERROR Request, and shall lead to the final state.

3.6.4.7 Requests and Indications

3.6.4.7.1 INVOKE

3.6.4.7.1.1 Function

3.6.4.7.1.1.1 The INVOKE Request primitive shall be used by the consumer application to initiate an INVOKE interaction.

3.6.4.7.1.1.2 The INVOKE Indication primitive shall be used by the provider MAL to deliver an INVOKE Message to a provider and initiate an INVOKE interaction.

3.6.4.7.1.2 Semantics

INVOKE Request and INVOKE Indication shall provide parameters as follows:

(INVOKE Message Header, INVOKE Message Body)

3.6.4.7.1.3 When Generated

3.6.4.7.1.3.1 An INVOKE Request may be generated by the consumer application at any time.

3.6.4.7.1.3.2 An INVOKE Indication shall be generated by the provider MAL upon reception of an INVOKE Message, once checked via the Access Control interface, from a consumer.

3.6.4.7.1.4 Effect on Reception

3.6.4.7.1.4.1 Reception of an INVOKE Request shall, once checked via the Access Control interface, cause the consumer MAL to initiate an INVOKE interaction by transmitting an INVOKE Message to the provider.

3.6.4.7.1.4.2 The consumer MAL shall enter the Initiated State.

3.6.4.7.1.4.3 On reception of an INVOKE Indication by the provider, the provider MAL shall enter the Initiated State.

3.6.4.7.1.4.4 At this point, the provider shall start processing the operation.

3.6.4.7.1.5 Message Header

3.6.4.7.1.5.1 For the INVOKE Message, the Message Header fields shall be as defined in 3.5 except for the fields in table 3-15.

3.6.4.7.1.5.2 The contents of the Message Body, as specified in the operation template, shall immediately follow the Message Header.

Table 3-15: INVOKE Message Header Fields

Field	Value
From	Consumer identifier
Authentication Id	Consumer Authentication Identifier
To	Provider identifier
Interaction Type	INVOKE
Interaction Stage	1
Transaction Id	Provided by consumer MAL

3.6.4.7.2 ACK

3.6.4.7.2.1 Function

3.6.4.7.2.1.1 The ACK Request primitive shall be used by the provider application to acknowledge an INVOKE interaction.

3.6.4.7.2.1.2 The ACK Indication primitive shall be used by the consumer MAL to deliver an ACK Message to a consumer.

3.6.4.7.2.2 Semantics

ACK Request and ACK Indication shall provide parameters as follows:

(ACK Message Header, ACK Message Body)

3.6.4.7.2.3 When Generated

3.6.4.7.2.3.1 An ACK Request shall be used by the provider application with the provider MAL in the Initiated State to acknowledge the INVOKE interaction.

3.6.4.7.2.3.2 An ACK Indication shall be generated by the consumer MAL in the Initiated State upon reception of an ACK Message, once checked via the Access Control interface, from a provider.

3.6.4.7.2.4 Effect on Reception

3.6.4.7.2.4.1 Reception of an ACK Request shall, once checked via the Access Control interface, cause the provider MAL to transmit the supplied acknowledgement as an ACK Message to the consumer.

3.6.4.7.2.4.2 A provider MAL in the Initiated State shall enter the Acknowledged State.

3.6.4.7.2.4.3 On reception of an ACK Indication by the consumer, the consumer MAL shall enter the Acknowledged State.

3.6.4.7.2.5 Message Header

3.6.4.7.2.5.1 For the ACK Message, the Message Header fields shall be as defined in 3.5 except for the fields in table 3-16.

3.6.4.7.2.5.2 The contents of the Message Body, as specified in the operation template, shall immediately follow the Message Header.

Table 3-16: Invoke ACK Message Header Fields

Field	Value
From	Provider identifier
Authentication Id	Provider Authentication Identifier
To	Consumer identifier
Interaction Type	INVOKE
Interaction Stage	2
Transaction Id	Transaction Id from initial message

3.6.4.7.3 ACK_ERROR

3.6.4.7.3.1 Function

3.6.4.7.3.1.1 The ACK_ERROR Request primitive shall be used by the provider application to end an INVOKE interaction with an error.

3.6.4.7.3.1.2 The ACK_ERROR Indication primitive shall be used by the consumer MAL to deliver an ACK_ERROR Message to a consumer.

3.6.4.7.3.2 Semantics

ACK_ERROR Request and ACK_ERROR Indication shall provide parameters as follows:

(ACK_ERROR Message Header, Error Number, Extra Information)

3.6.4.7.3.3 When Generated

3.6.4.7.3.3.1 An ACK_ERROR Request shall be used by the provider application with the provider MAL in the Initiated State to transmit an error.

3.6.4.7.3.3.2 An ACK_ERROR Indication shall be generated by the consumer MAL in the Initiated State upon one of two events:

a) the reception of an ACK_ERROR Message, once checked via the Access Control interface, from a provider;

b) an error raised by the local communication layer.

3.6.4.7.3.4 Effect on Reception

3.6.4.7.3.4.1 Reception of an ACK_ERROR Request shall, once checked via the Access Control interface, cause the provider MAL to transmit an ACK_ERROR Message to the consumer.

3.6.4.7.3.4.2 The pattern shall end at this point for the provider.

3.6.4.7.3.4.3 On reception of an ACK_ERROR Indication, a consumer shall end the interaction with an error.

3.6.4.7.3.5 Message Header

3.6.4.7.3.5.1 For the ACK_ERROR Message, the Message Header fields shall be the same as for the ACK Message except for the Is Error Message field, which is set to TRUE.

3.6.4.7.3.5.2 The Error Information shall immediately follow the Message Header.

3.6.4.7.4 RESPONSE

3.6.4.7.4.1 Function

3.6.4.7.4.1.1 The RESPONSE Request primitive shall be used by the provider application to transmit a response to an INVOKE interaction.

3.6.4.7.4.1.2 The RESPONSE Indication primitive shall be used by the consumer MAL to deliver a RESPONSE Message to a consumer.

3.6.4.7.4.2 Semantics

RESPONSE Request and RESPONSE Indication shall provide parameters as follows:

(RESPONSE Message Header, RESPONSE Message Body)

3.6.4.7.4.3 When Generated

3.6.4.7.4.3.1 A RESPONSE Request shall be used by the provider application with the provider MAL in the Acknowledged State to transmit a final response to an INVOKE interaction.

3.6.4.7.4.3.2 A RESPONSE Indication shall be generated by the consumer MAL in the Acknowledged State upon reception of a RESPONSE Message, once checked via the Access Control interface, from a provider.

3.6.4.7.4.4 Effect on Reception

3.6.4.7.4.4.1 Reception of a RESPONSE Request shall, once checked via the Access Control interface, cause the provider MAL to transmit the supplied response as a RESPONSE Message to the consumer.

3.6.4.7.4.4.2 The pattern shall end at this point for the provider.

3.6.4.7.4.4.3 On reception of a RESPONSE Indication, a consumer shall end the INVOKE interaction with success.

3.6.4.7.4.5 Message Header

3.6.4.7.4.5.1 For the RESPONSE Message, the Message Header fields shall be as defined in 3.5 except for the fields in table 3-17.

3.6.4.7.4.5.2 The contents of the Message Body, as specified in the operation template, shall immediately follow the Message Header.

Table 3-17: Invoke RESPONSE Message Header Fields

Field	Value
From	Provider identifier
Authentication Id	Provider Authentication Identifier
To	Consumer identifier
Interaction Type	INVOKE
Interaction Stage	3
Transaction Id	Transaction Id from initial message

3.6.4.7.5 RESPONSE_ERROR

3.6.4.7.5.1 Function

3.6.4.7.5.1.1 The RESPONSE_ERROR Request primitive shall be used by the provider application to end an INVOKE interaction with an error.

3.6.4.7.5.1.2 The RESPONSE_ERROR Indication primitive shall be used by the consumer MAL to deliver a RESPONSE_ERROR Message to a consumer.

3.6.4.7.5.2 Semantics

RESPONSE_ERROR Request and RESPONSE_ERROR Indication shall provide parameters as follows:

(RESPONSE_ERROR Message Header, Error Number, Extra Information)

3.6.4.7.5.3 When Generated

3.6.4.7.5.3.1 A RESPONSE_ERROR Request shall be used by the provider application with the provider MAL in the Acknowledged State to transmit an error.

3.6.4.7.5.3.2 A RESPONSE_ERROR Indication shall be generated by the consumer MAL in the Acknowledged State upon one of two events:

a) the reception of a RESPONSE_ERROR Message, once checked via the Access Control interface, from a provider;

b) an error raised by the local communication layer.

3.6.4.7.5.4 Effect on Reception

3.6.4.7.5.4.1 Reception of a RESPONSE_ERROR Request shall, once checked via the Access Control interface, cause the provider MAL to transmit a RESPONSE_ERROR Message to the consumer.

3.6.4.7.5.4.2 The pattern shall end at this point for the provider.

3.6.4.7.5.4.3 On reception of a RESPONSE_ERROR Indication, a consumer shall end the interaction with an error.

3.6.4.7.5.5 Message Header

3.6.4.7.5.5.1 For the RESPONSE_ERROR Message, the Message Header fields shall be the same as for the RESPONSE Message except for the Is Error Message field, which is set to TRUE.

3.6.4.7.5.5.2 The Error Information shall immediately follow the Message Header.

3.6.4.8 Discussion

The following example shows a simple example service that contains a single INVOKE operation. The operation sends a TestBody structure and returns a TestAck acknowledgment structure followed by a TestResponse structure:

Operation Identifier	testInvoke		
Interaction Pattern	INVOKE		
Pattern Sequence	Message	Nullable	Type Signature
IN	INVOKE	Yes	TestBody body
OUT	ACK	Yes	TestAck ack
OUT	RESPONSE	Yes	TestResponse response

The TestBody structure is defined in 3.6.1.8, and the TestResponse structure is defined in 3.6.3.8. The TestAck structure is defined below:

Name	TestAck		
Extends	Composite		
Short Form Part	Example Only		
Field	Type	Nullable	Comment
AckId	Identifier	Yes	Example Identifier item

This corresponds to the following message being transmitted:

Message Header													TestBody	
From	Authentication Id	To	Timestamp	Interaction Type	Interaction Stage	Transaction Id	Service Area	Service	Operation	Area Version	Is Error Message	Supplements	FirstItem	SecondItem
Consumer	Op A	Provider		INVOKE	1	123	Example	Example	testInvoke	1	FALSE		Hello	1

And it corresponds to the following acknowledgement being returned:

Message Header													TestAck
From	Authentication Id	To	Timestamp	Interaction Type	Interaction Stage	Transaction Id	Service Area	Service	Operation	Area Version	Is Error Message	Supplements	AckId
Provider	SC X	Consumer		INVOKE	2	123	Example	Example	testInvoke	1	FALSE		123

And finally, it corresponds to the following response being returned:

Message Header													TestResponse	
From	Authentication Id	To	Timestamp	Interaction Type	Interaction Stage	Transaction Id	Service Area	Service	Operation	Area Version	Is Error Message	Supplements	RspnItem	RspnField
Provider	SC X	Consumer		INVOKE	3	123	Example	Example	testInvoke	1	FALSE		TRUE	31.0

NOTE – An actual service specification, rather than the example given here, would fully specify the meaning of, and required values of, all structures used by the service.

3.6.5 PROGRESS INTERACTION PATTERN

3.6.5.1 Overview

3.6.5.1.1 General

The PROGRESS pattern extends the INVOKE pattern by adding a set of progress updates between the acknowledgement message and the response message (figure 3-16). This allows the operation to confirm the receipt of the Request before proceeding to process the Request and also to return multiple update messages for that Request.

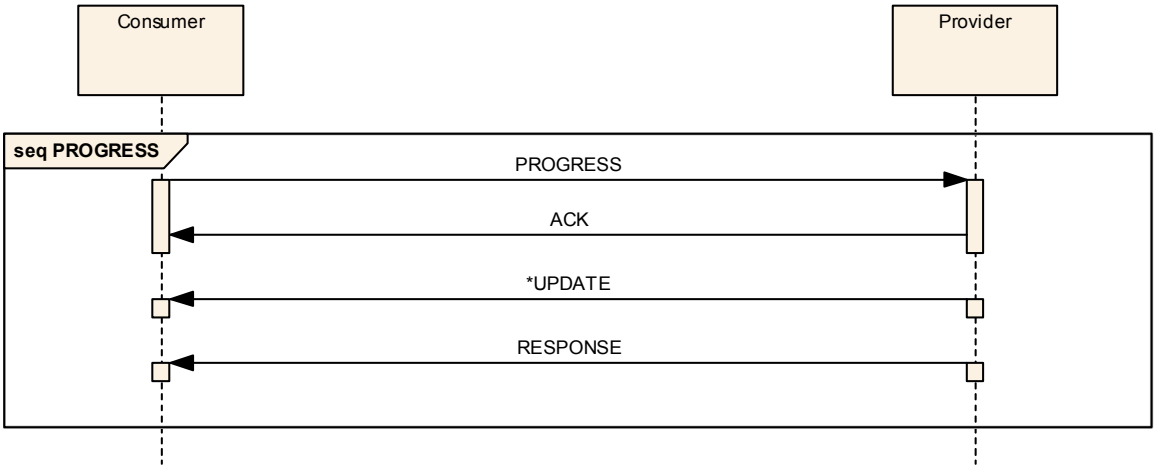

Figure 3-16: PROGRESS Interaction Pattern Message Sequence

3.6.5.1.2 Description

The PROGRESS pattern extends the INVOKE pattern with the addition of a set of mandatory progress messages. The type of progress messages and their number are defined by the service and not by the pattern.

3.6.5.2 Usage

The PROGRESS pattern should be used when there is a significant or indeterminate amount of time taken to process the Request and produce the data response, and where monitoring of the progress of the operation is required or the data response is to be returned in blocks. The order of the update messages is not guaranteed by the MAL; thus it is delegated to the Transport Layer on an implementation-specific basis (e.g., with a MAL-TCP/IP transport binding).

NOTE – The order of the update messages cannot be guaranteed at the MAL level due to the technical fact that none of the update messages is stamped with an index. Thus, the receiving MAL would not be able to sort the messages correctly in case they are out-of-order.

3.6.5.3 Error Handling

The PROGRESS pattern may report failure in three distinct ways:

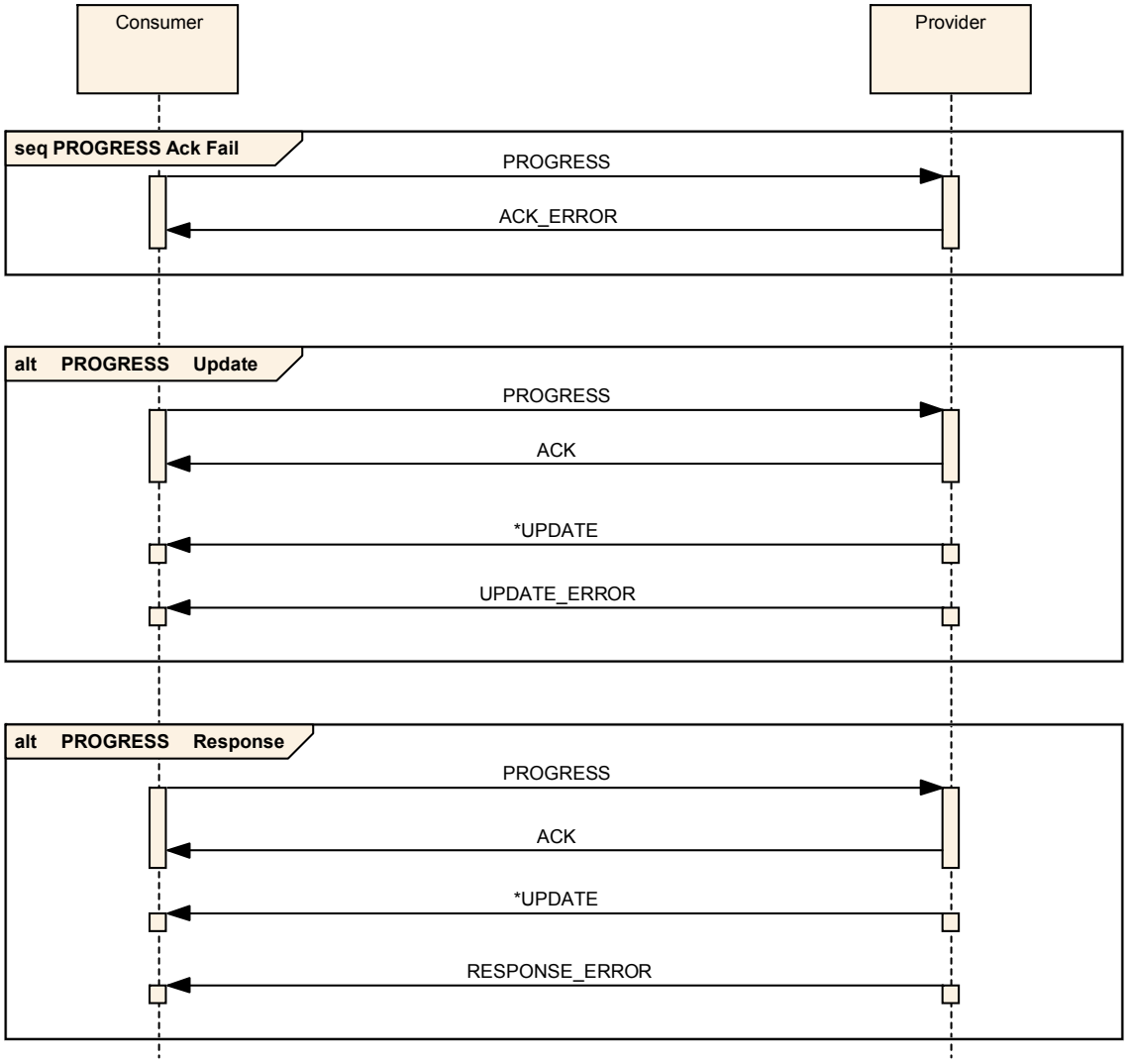

Figure 3-17: PROGRESS Interaction Pattern Error Sequence

a) The acknowledgement may be replaced with an error message (see section 4) shown in the first sequence in figure 3-17.

b) An error may be generated during the processing of the body of the operation after the initial acknowledgement is sent, as shown in the second sequence in figure 3-17, and will replace any of the progress updates.

c) An error may be generated after the updates and will replace the final response as shown in the third sequence in figure 3-17.

d) After an error message is sent, no further messages shall be generated as part of the pattern.

3.6.5.4 Operation Template

The PROGRESS pattern shall conform to the PROGRESS operation template defined in table 3-18.

Table 3-18: PROGRESS Operation Template

Operation Identifier	<<Operation name>>		
Interaction Pattern	PROGRESS		
Pattern Sequence	Message	Nullable	Type Signature
IN	PROGRESS	<<Nullability>>	<<Progress Signature>>
OUT	ACK	<<Nullability>>	<<Ack Signature>>
OUT	UPDATE	<<Nullability>>	<<Update Signature>>
OUT	RESPONSE	<<Nullability>>	<<Response Signature>>

NOTE – The PROGRESS template is similar to the INVOKE pattern template but adds a progress update message.

3.6.5.5 Primitives

The PROGRESS pattern shall use the primitives defined in table 3-19.

Table 3-19: PROGRESS Primitive List

Primitive
PROGRESS
ACK
ACK_ERROR
UPDATE
UPDATE_ERROR
RESPONSE
RESPONSE_ERROR

3.6.5.6 State Charts

3.6.5.6.1 Consumer Side

3.6.5.6.1.1 Figure 3-18 is the consumer side state chart for the PROGRESS pattern.

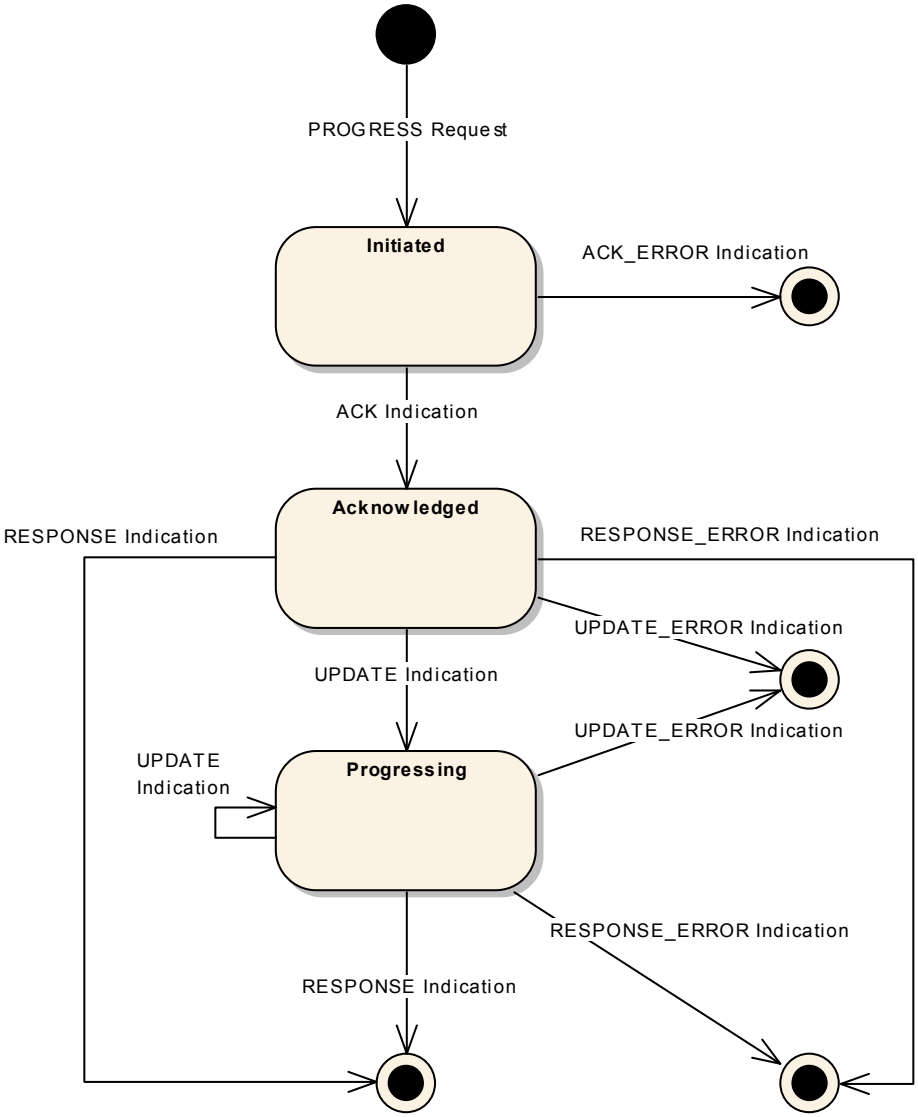

Figure 3-18: PROGRESS Consumer State Chart

3.6.5.6.1.2 The initial transition shall be triggered by the primitive PROGRESS Request and shall lead to the Initiated State.

3.6.5.6.1.3 There are two possible transitions from the Initiated State:

a) the first is the indication of an acknowledgement, ACK Indication, and shall lead to the Acknowledged State;

b) the second is the indication of an error, ACK_ERROR Indication, and shall lead to the final state.

3.6.5.6.1.4 There are four possible transitions from the Acknowledged State:

a) the first is the indication of an update, UPDATE Indication, and shall lead to the Progressing State;

b) the second is the indication of an error in generating an initial update, UPDATE_ERROR Indication, and shall lead to the final state;

c) the third is the indication of a response, RESPONSE Indication, and shall lead to the final state;

d) the fourth is the indication of an error in generating the response, RESPONSE_ERROR Indication, and shall lead to the final state.

3.6.5.6.1.5 There are four possible transitions from the Progressing State:

a) the first is the indication of another update, UPDATE Indication, and shall lead back to the Progressing State;

b) the second is the indication of an error in generating an update, UPDATE_ERROR Indication, and shall lead to the final state;

c) the third is the indication of a response, RESPONSE Indication, and shall lead to the final state;

d) the fourth is the indication of an error in generating the response, RESPONSE_ERROR Indication, and shall lead to the final state.

3.6.5.6.2 Provider Side

3.6.5.6.2.1 Figure 3-19 is the provider side state chart for the PROGRESS pattern.

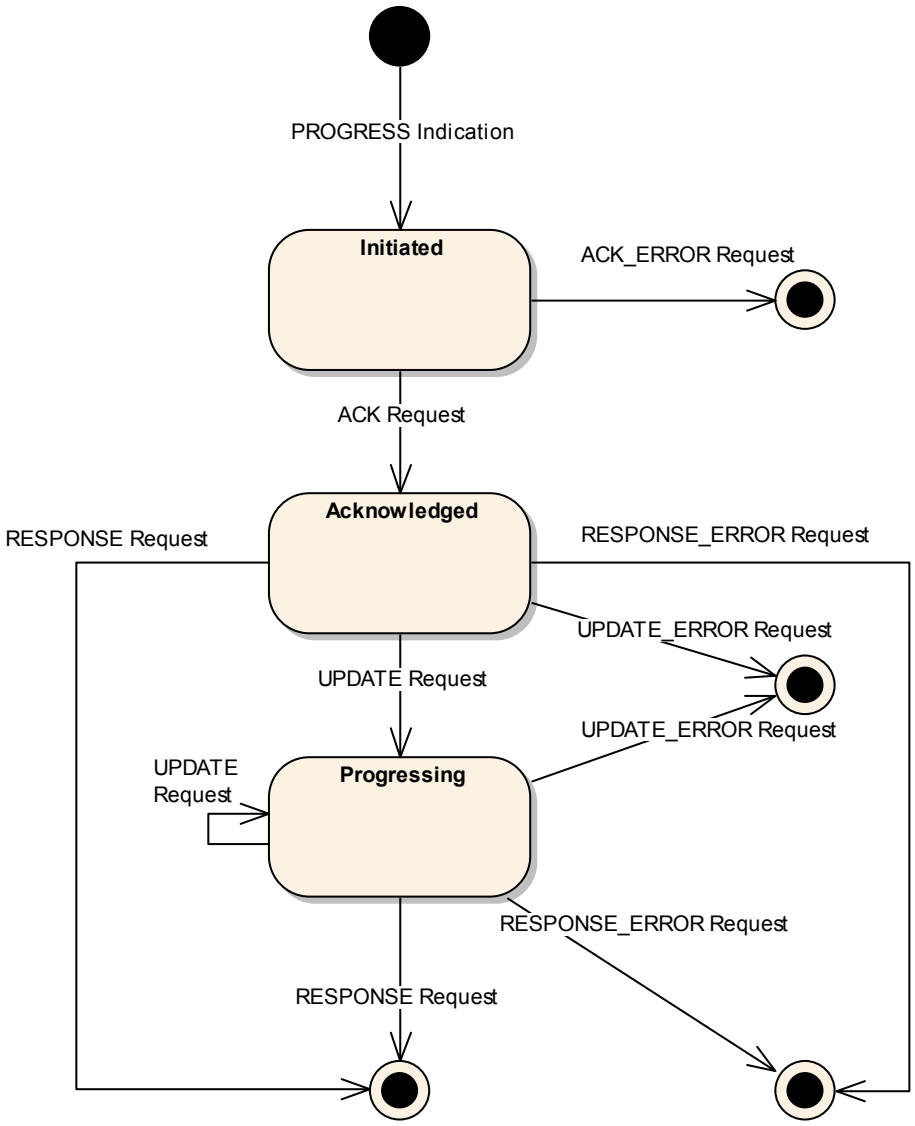

Figure 3-19: PROGRESS Provider State Chart

3.6.5.6.2.2 The initial transition shall be triggered by the primitive PROGRESS Indication and shall lead to the Initiated State.

3.6.5.6.2.3 There are two possible transitions from the Initiated State:

a) the first is the transmission of an acknowledgement, ACK Request, and shall lead to the Acknowledged State;

b) the second is the transmission of an error, ACK_ERROR Request, and shall lead to the final state.

3.6.5.6.2.4 There are four possible transitions from the Acknowledged State:

a) the first is the transmission of an update, UPDATE Request, and shall lead to the Progressing State;

b) the second is the transmission of an error in generating an initial update, UPDATE_ERROR Request, and shall lead to the final state;

c) the third is the transmission of a response, RESPONSE Request, and shall lead to the final state;

d) the fourth is the transmission of an error in generating the response, RESPONSE_ERROR Request, and shall lead to the final state.

3.6.5.6.2.5 There are four possible transitions from the Progressing State:

a) the first is the transmission of another update, UPDATE Request, and shall lead back to the Progressing State;

b) the second is the transmission of an error in generating an update, UPDATE_ERROR Request, and shall lead to the final state;

c) the third is the transmission of a response, RESPONSE Request, and shall lead to the final state;

d) the fourth is the transmission of an error in generating the response, RESPONSE_ERROR Request, and shall lead to the final state.

3.6.5.7 Requests and Indications

3.6.5.7.1 PROGRESS

3.6.5.7.1.1 Function

3.6.5.7.1.1.1 The PROGRESS Request primitive shall be used by the consumer application to initiate a PROGRESS interaction.

3.6.5.7.1.1.2 The PROGRESS Indication primitive shall be used by the provider MAL to deliver a PROGRESS Message to a provider and initiate a PROGRESS interaction.

3.6.5.7.1.2 Semantics

PROGRESS Request and PROGRESS Indication shall provide parameters as follows:

(PROGRESS Message Header, PROGRESS Message Body)

3.6.5.7.1.3 When Generated

3.6.5.7.1.3.1 A PROGRESS Request may be generated by the consumer application at any time.

3.6.5.7.1.3.2 A PROGRESS Indication shall be generated by the provider MAL upon reception of a PROGRESS Message, once checked via the Access Control interface, from a consumer.

3.6.5.7.1.4 Effect on Reception

3.6.5.7.1.4.1 Reception of a PROGRESS Request shall, once checked via the Access Control interface, cause the consumer MAL to initiate a PROGRESS interaction by transmitting a PROGRESS Message to the provider.

3.6.5.7.1.4.2 The consumer MAL shall enter the Initiated State.

3.6.5.7.1.4.3 On reception of a PROGRESS Indication by a provider, the provider MAL shall enter the Initiated State.

3.6.5.7.1.4.4 At this point, the provider shall start processing the operation.

3.6.5.7.1.5 Message Header

3.6.5.7.1.5.1 For the PROGRESS Message, the Message Header fields shall be as defined in 3.5 except for the fields in table 3-20.

3.6.5.7.1.5.2 The contents of the Message Body, as specified in the operation template, shall immediately follow the Message Header.

Table 3-20: PROGRESS Message Header Fields

Field	Value
From	Consumer identifier
Authentication Id	Consumer Authentication Identifier
To	Provider identifier
Interaction Type	PROGRESS
Interaction Stage	1
Transaction Id	Provided by consumer MAL

3.6.5.7.2 ACK

3.6.5.7.2.1 Function

3.6.5.7.2.1.1 The ACK Request primitive shall be used by the provider application to acknowledge a PROGRESS interaction.

3.6.5.7.2.1.2 The ACK Indication primitive shall be used by the consumer MAL to deliver an ACK Message to a consumer.

3.6.5.7.2.2 Semantics

ACK Request and ACK Indication shall provide parameters as follows:

(ACK Message Header, ACK Message Body)

3.6.5.7.2.3 When Generated

3.6.5.7.2.3.1 An ACK Request shall be used by the provider application with the provider MAL in the Initiated State to acknowledge a PROGRESS interaction.

3.6.5.7.2.3.2 An ACK Indication shall be generated by the consumer MAL in the Initiated State upon reception of an ACK Message, once checked via the Access Control interface, from a provider.

3.6.5.7.2.4 Effect on Reception

3.6.5.7.2.4.1 Reception of an ACK Request shall, once checked via the Access Control interface, cause the provider MAL to transmit the supplied acknowledgement as an ACK Message to the consumer.

3.6.5.7.2.4.2 A provider MAL in the Initiated State shall enter the Acknowledged State.

3.6.5.7.2.4.3 On reception of an ACK Indication by the consumer, the consumer MAL shall enter the Acknowledged State.

3.6.5.7.2.5 Message Header

3.6.5.7.2.5.1 For the ACK Message, the Message Header fields shall be as defined in 3.5 except for the fields in table 3-21.

3.6.5.7.2.5.2 The contents of the Message Body, as specified in the operation template, shall immediately follow the Message Header.

Table 3-21: Progress ACK Message Header Fields

Field	Value
From	Provider identifier
Authentication Id	Provider Authentication Identifier
To	Consumer identifier
Interaction Type	PROGRESS
Interaction Stage	2
Transaction Id	Transaction Id from initial message

3.6.5.7.3 ACK_ERROR

3.6.5.7.3.1 Function

3.6.5.7.3.1.1 The ACK_ERROR Request primitive shall be used by the provider application to end a PROGRESS interaction with an error.

3.6.5.7.3.1.2 The ACK_ERROR Indication primitive shall be used by the consumer MAL to deliver an ACK_ERROR Message to a consumer.

3.6.5.7.3.2 Semantics

ACK_ERROR Request and ACK_ERROR Indication shall provide parameters as follows:

(ACK_ERROR Message Header, Error Number, Extra Information)

3.6.5.7.3.3 When Generated

3.6.5.7.3.3.1 An ACK_ERROR Request shall be used by the provider application with the provider MAL in the Initiated State to transmit an error.

3.6.5.7.3.3.2 An ACK_ERROR Indication shall be generated by the consumer MAL in the Initiated State upon one of two events:

a) the reception of an ACK_ERROR Message, once checked via the Access Control interface, from a provider;

b) an error raised by the local communication layer.

3.6.5.7.3.4 Effect on Reception

3.6.5.7.3.4.1 Reception of an ACK_ERROR Request shall, once checked via the Access Control interface, cause the provider MAL to transmit an ACK_ERROR Message to the consumer.

3.6.5.7.3.4.2 The pattern shall end at this point for the provider.

3.6.5.7.3.4.3 On reception of an ACK_ERROR Indication, a consumer shall end the interaction with an error.

3.6.5.7.3.5 Message Header

3.6.5.7.3.5.1 For the ACK_ERROR Message, the Message Header fields shall be the same as for the ACK Message except for the Is Error Message field, which is set to TRUE.

3.6.5.7.3.5.2 The Error Information shall immediately follow the Message Header.

3.6.5.7.4 UPDATE

3.6.5.7.4.1 Function

3.6.5.7.4.1.1 The UPDATE Request primitive shall be used by the provider application to transmit an update during a PROGRESS interaction.

3.6.5.7.4.1.2 The UPDATE Indication primitive shall be used by the consumer MAL to deliver an UPDATE Message to a consumer.

3.6.5.7.4.2 Semantics

UPDATE Request and UPDATE Indication shall provide parameters as follows:

(UPDATE Message Header, UPDATE Message Body)

3.6.5.7.4.3 When Generated

3.6.5.7.4.3.1 An UPDATE Request shall be used by the provider application with the provider MAL in either the Acknowledged State or Progressing State to transmit an update to a PROGRESS interaction.

3.6.5.7.4.3.2 An UPDATE Indication shall be generated by the consumer MAL in either the Acknowledged State or the Progressing State upon reception of an UPDATE Message, once checked via the Access Control interface, from a provider.

3.6.5.7.4.4 Effect on Reception

3.6.5.7.4.4.1 Reception of an UPDATE Request shall, once checked via the Access Control interface, cause the provider MAL to transmit the supplied update as an UPDATE Message to the consumer.

3.6.5.7.4.4.2 A provider MAL in the Acknowledged State shall enter the Progressing State.

3.6.5.7.4.4.3 On reception of an UPDATE Indication by the consumer, the consumer MAL in the Acknowledged State shall enter the Progressing State.

3.6.5.7.4.5 Message Header

3.6.5.7.4.5.1 For the UPDATE Message, of which there may be several, the Message Header fields shall be as defined in 3.5 except for the fields in table 3-22.

3.6.5.7.4.5.2 The contents of the Message Body, as specified in the operation template, shall immediately follow the Message Header.

Table 3-22: Progress UPDATE Message Header Fields

Field	Value
From	Provider identifier
Authentication Id	Provider Authentication Identifier
To	Consumer identifier
Interaction Type	PROGRESS
Interaction Stage	3
Transaction Id	Transaction Id from initial message

3.6.5.7.5 UPDATE_ERROR

3.6.5.7.5.1 Function

3.6.5.7.5.1.1 The UPDATE_ERROR Request primitive shall be used by the provider application to end a PROGRESS interaction with an error.

3.6.5.7.5.1.2 The UPDATE_ERROR Indication primitive shall be used by the consumer MAL to deliver an UPDATE_ERROR Message to a consumer.

3.6.5.7.5.2 Semantics

UPDATE_ERROR Request and UPDATE_ERROR Indication shall provide parameters as follows:

(UPDATE_ERROR Message Header, Error Number, Extra Information)

3.6.5.7.5.3 When Generated

3.6.5.7.5.3.1 An UPDATE_ERROR Request shall be used by the provider application with the provider MAL in either the Acknowledged State or the Progressing State to transmit an error.

3.6.5.7.5.3.2 An UPDATE_ERROR Indication shall be generated by the consumer MAL in either the Acknowledged State or the Progressing State upon one of two events:

a) the reception of an UPDATE_ERROR Message, once checked via the Access Control interface, from a provider;

b) an error raised by the local communication layer.

3.6.5.7.5.4 Effect on Reception

3.6.5.7.5.4.1 Reception of an UPDATE_ERROR Request shall, once checked via the Access Control interface, cause the provider MAL to transmit an UPDATE_ERROR Message to the consumer.

3.6.5.7.5.4.2 The pattern shall end at this point for the provider.

3.6.5.7.5.4.3 On reception of an UPDATE_ERROR Indication, a consumer shall end the interaction with an error.

3.6.5.7.5.5 Message Header

3.6.5.7.5.5.1 For the UPDATE_ERROR Message, the Message Header fields shall be the same as for the UPDATE Message except for the Is Error Message field, which is set to TRUE.

3.6.5.7.5.5.2 The Error Information shall immediately follow the Message Header.

3.6.5.7.6 RESPONSE

3.6.5.7.6.1 Function

3.6.5.7.6.1.1 The RESPONSE Request primitive shall be used by the provider application to transmit a response to a PROGRESS interaction.

3.6.5.7.6.1.2 The RESPONSE Indication primitive shall be used by the consumer MAL to deliver a RESPONSE Message to a consumer.

3.6.5.7.6.2 Semantics

RESPONSE Request and RESPONSE Indication shall provide parameters as follows:

(RESPONSE Message Header, RESPONSE Message Body)

3.6.5.7.6.3 When Generated

3.6.5.7.6.3.1 A RESPONSE Request shall be used by the provider application with the provider MAL in either the Acknowledged State or Progressing State to transmit a final response to a PROGRESS interaction.

3.6.5.7.6.3.2 A RESPONSE Indication shall be generated by the consumer MAL in either the Acknowledged State or the Progressing State upon reception of a RESPONSE Message, once checked via the Access Control interface, from a provider.

3.6.5.7.6.4 Effect on Reception

3.6.5.7.6.4.1 Reception of a RESPONSE Request shall, once checked via the Access Control interface, cause the provider MAL to transmit the supplied response as a RESPONSE Message to the consumer.

3.6.5.7.6.4.2 The pattern shall end at this point for the provider.

3.6.5.7.6.4.3 On reception of a RESPONSE Indication, a consumer shall end the PROGRESS interaction with success.

3.6.5.7.6.5 Message Header

3.6.5.7.6.5.1 For the RESPONSE Message, the Message Header fields shall be as defined in 3.5 except for the fields in table 3-23.

3.6.5.7.6.5.2 The contents of the Message Body, as specified in the operation template, shall immediately follow the Message Header.

Table 3-23: Progress RESPONSE Message Header Fields

Field	Value
From	Provider identifier
Authentication Id	Provider Authentication Identifier
To	Consumer identifier
Interaction Type	PROGRESS
Interaction Stage	4
Transaction Id	Transaction Id from initial message

3.6.5.7.7 RESPONSE_ERROR

3.6.5.7.7.1 Function

3.6.5.7.7.1.1 The RESPONSE_ERROR Request primitive shall be used by the provider application to end a PROGRESS interaction with an error.

3.6.5.7.7.1.2 The RESPONSE_ERROR Indication primitive shall be used by the consumer MAL to deliver a RESPONSE_ERROR Message to a consumer.

3.6.5.7.7.2 Semantics

RESPONSE_ERROR Request and RESPONSE_ERROR Indication shall provide parameters as follows:

(RESPONSE_ERROR Message Header, Error Number, Extra Information)

3.6.5.7.7.3 When Generated

3.6.5.7.7.3.1 A RESPONSE_ERROR Request shall be used by the provider application with the provider MAL in either the Acknowledged State or Progressing State to transmit an error.

3.6.5.7.7.3.2 A RESPONSE_ERROR Indication shall be generated by the consumer MAL in either the Acknowledged State or the Progressing State upon one of two events:

a) the reception of a RESPONSE_ERROR Message, once checked via the Access Control interface, from a provider;

b) an error raised by the local communication layer.

3.6.5.7.7.4 Effect on Reception

3.6.5.7.7.4.1 Reception of a RESPONSE_ERROR Request shall, once checked via the Access Control interface, cause the provider MAL to transmit a RESPONSE_ERROR Message to the consumer.

3.6.5.7.7.4.2 The pattern shall end at this point for the provider.

3.6.5.7.7.4.3 On reception of a RESPONSE_ERROR Indication, a consumer shall end the interaction with an error.

3.6.5.7.7.5 Message Header

3.6.5.7.7.5.1 For the RESPONSE_ERROR Message, the Message Header fields shall be the same as for the RESPONSE Message except for the Is Error Message field, which is set to TRUE.

3.6.5.7.7.5.2 The Error Information shall immediately follow the Message Header.

3.6.5.8 Discussion

The following example shows a simple example service that contains a single PROGRESS operation. The operation sends a TestBody structure, which is acknowledged with a TestAck structure, reports progress with a TestProgress structure, and returns a TestResponse structure:

Operation Identifier	testProgress		
Interaction Pattern	PROGRESS		
Pattern Sequence	Message	Nullable	Type Signature
IN	PROGRESS	Yes	TestBody body
OUT	ACK	Yes	TestAck ack
OUT	UPDATE	Yes	TestProgress progress
OUT	RESPONSE	Yes	TestResponse response

The TestBody structure is defined in 3.6.1.8, TestAck structure is defined in 3.6.4.8, and the TestResponse structure is defined in 3.6.3.8. The TestProgress structure is defined below:

Name	TestProgress		
Extends	Composite		
Short Form Part	Example Only		
Field	Type	Nullable	Comment
RspnItem	Integer	Yes	Example Integer item

This corresponds to the following message being transmitted:

Message Header													TestBody	
From	Authentication Id	To	Timestamp	Interaction Type	Interaction Stage	Transaction Id	Service Area	Service	Operation	Area Version	Is Error Message	Supplements	FirstItem	SecondItem
Consumer	Op A	Provider		PROGRESS	1	123	Example	Example	testProg.	1	FALSE		Hello	1

And it corresponds to the following acknowledgement being returned:

Message Header													TestAck
From	Authentication Id	To	Timestamp	Interaction Type	Interaction Stage	Transaction Id	Service Area	Service	Operation	Area Version	Is Error Message	Supplements	AckId
Provider	SC X	Consumer		PROGRESS	2	123	Example	Example	testProg.	1	FALSE		1234

And it corresponds, assuming an example with two updates, to the following being returned:

Message Header													TestProgress
From	Authentication Id	To	Timestamp	Interaction Type	Interaction Stage	Transaction Id	Service Area	Service	Operation	Area Version	Is Error Message	Supplements	RspnItem
Provider	SC X	Consumer		PROGRESS	3	123	Example	Example	testProg.	1	FALSE		1

Message Header													TestProgress
From	Authentication Id	To	Timestamp	Interaction Type	Interaction Stage	Transaction Id	Service Area	Service	Operation	Area Version	Is Error Message	Supplements	RspnItem
Provider	SC X	Consumer		PROGRESS	3	123	Example	Example	testProg.	1	FALSE		2

And it finally corresponds to the following response being returned:

Message Header													TestResponse	
From	Authentication Id	To	Timestamp	Interaction Type	Interaction Stage	Transaction Id	Service Area	Service	Operation	Area Version	Is Error Message	Supplements	RspnItem	RspnField
Provider	SC X	Consumer		PROGRESS	4	123	Example	Example	testProg.	1	FALSE		TRUE	31.0

NOTE – An actual service specification, rather than the example given here, would fully specify the meaning of, and required values of, all structures used by the service.

3.6.6 PUBLISH-SUBSCRIBE INTERACTION PATTERN

3.6.6.1 Overview

3.6.6.1.1 General

The PUBLISH-SUBSCRIBE pattern provides the ability for consumers to register interest in a specific topic and receive updates from one or more providers without requiring awareness of the number of, and location of, these providers.

The basic outline of the pattern is: 1) consumers register interest by sending a subscription list; 2) providers register on the broker; 3) providers publish updates; 4) updates are then sent to registered consumers; 5) when no further updates are required, the consumer cancels its subscription by deregistering.

The PUBLISH-SUBSCRIBE pattern can be deployed in two distinct ways. In the first deployment, an intermediary shared message broker resides between the consumers and the providers (figure 3-20). The shared broker permits a decoupling of the consumer from the providers, receiving the subscriptions from consumers and the updates from providers, and is responsible for dispatching the required updates to registered consumers:

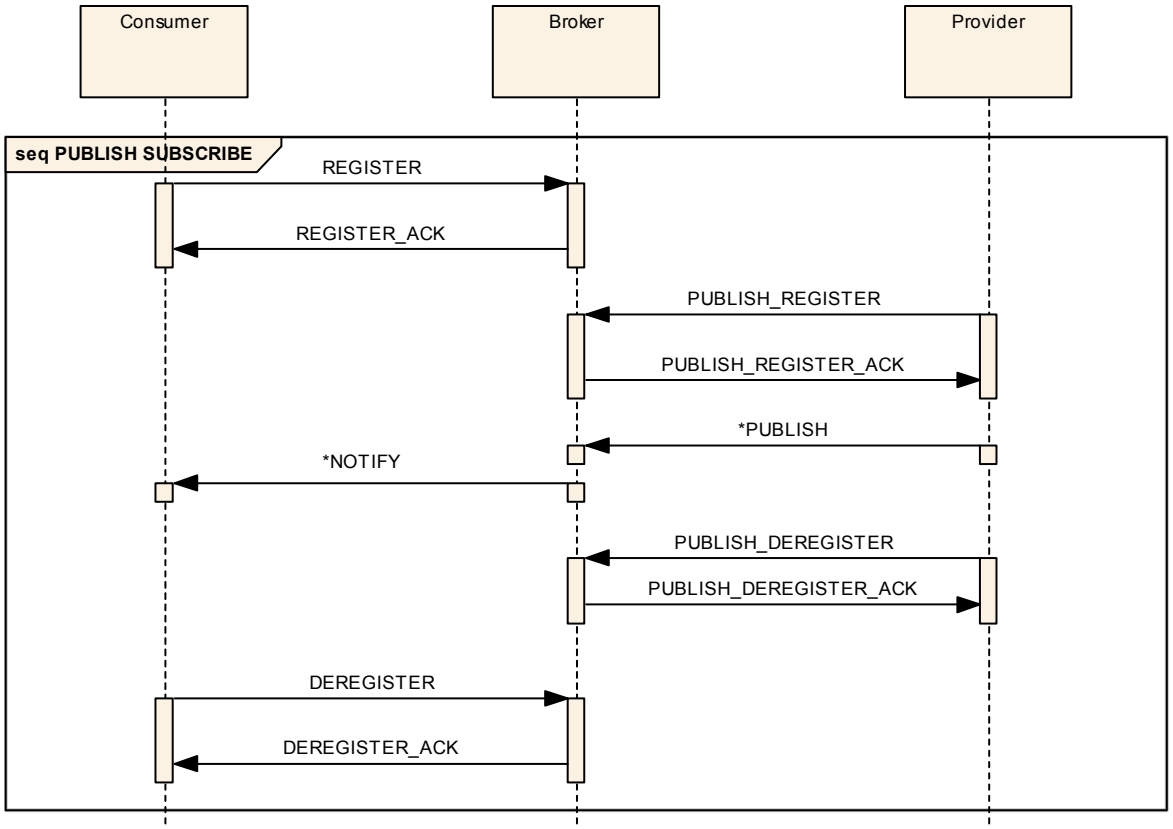

Figure 3-20: PUBLISH-SUBSCRIBE Interaction Pattern Message Sequence

The shared message broker manages the consumer subscription lists; the providers are not aware of what consumers there are and what entities they require. Because of this, they must publish all items they are configured to generate to the broker so that it can manage the updates to the consumers.

At a later time, when updates are no longer required, the consumer can deregister for these updates from the message broker.

The broker acknowledges both the consumer and provider register/deregister operations; these are treated like normal SUBMIT Messages. No acknowledgement of the PUBLISH Message is provided by the broker, and no acknowledgement by the consumer of the NOTIFY Message is required.

In the second deployment of the PUBLISH-SUBSCRIBE pattern, there is a single update provider, to which the message broker is private. This is an example of an observe type relationship in which a consumer observes some aspect of a specific provider.

In this deployment, because of the one-to-one relationship between the consumer and the provider, the functionality of the message broker does not necessarily require a separate entity. It is completely possible for the message broker logic to reside in the provider application; the provider publishes updates to itself for distribution to consumers (figure 3-21):

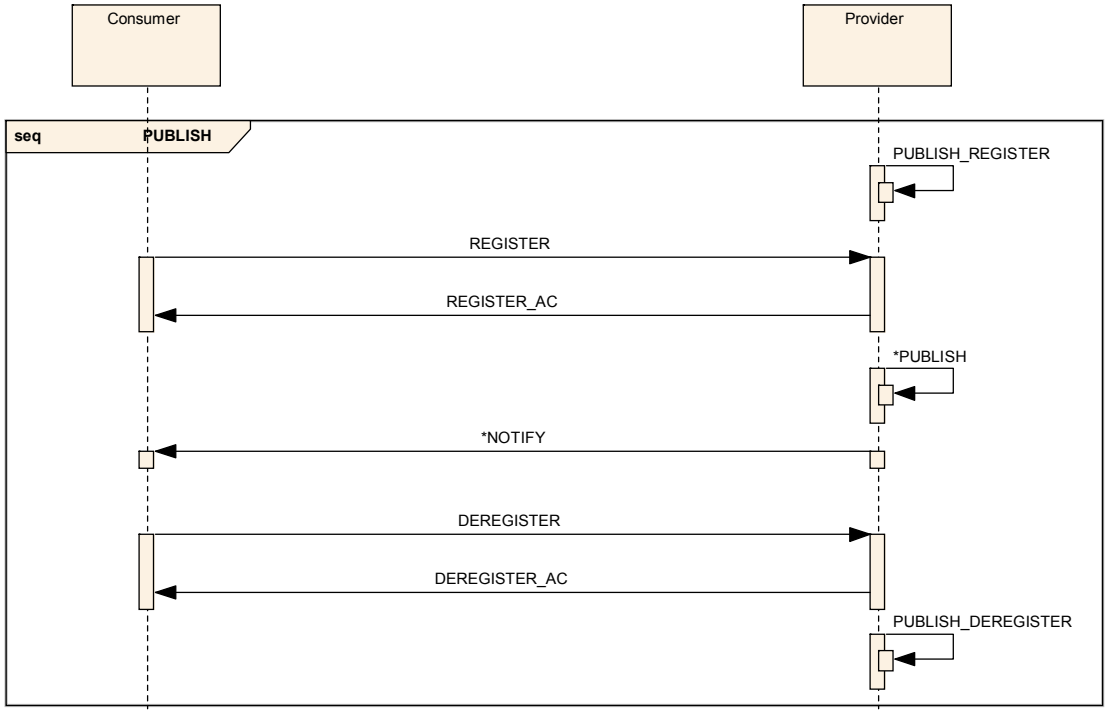

Figure 3-21: PUBLISH-SUBSCRIBE Pattern Alternative Message Sequence

However, because the pattern is still the same, all features of the first deployment (multiple concurrent subscription lists) are still supported.

In terms of implementation, for both the shared or private broker pattern, there are three primary implementation designs (however, others may exist). In the first design, the broker logic resides in the MAL layer; in the second, it is natively supported by the message transport technology used; and in the third, it is implemented as a standalone component external to the MAL. In all three cases, the pattern of interaction is the same; it is only the physical location of the broker that changes.

3.6.6.1.2 Description

The PUBLISH-SUBSCRIBE pattern for both the consumer and provider register/deregister has a predefined pattern message structure which allows an implementation of the message broker to manage the mapping of consumers to updates and hide this complexity from Provider applications.

This mapping of provider updates to consumer subscriptions is based on the concept of subscriptions keys. A Publish-Subscribe operation defines a set of Subscription Keys in its operation template. At runtime, the consumers can register with a set of filters based on some of the Subscription Keys, while the provider publishes Updates with values for each Subscription Key. The broker matches the Update key values with the filters to forward the message to the right consumers.

3.6.6.2 Broker—Behaviour

3.6.6.2.1 A consumer shall use subscriptions in order to inform the broker of the updates that it intends to receive. Subscriptions shall be sent during the registration.

3.6.6.2.2 A single consumer may define many subscriptions concurrently by specifying a different subscription identifier during the registration.

3.6.6.2.3 To change the contents of a subscription list, a consumer may redefine it by registering a new list with the same identifier. The consumer is not required to deregister the subscription first.

3.6.6.2.4 For a broker the replacement of a subscription is an atomic operation; no updates shall be missed as a result of the subscription replacement operation.

3.6.6.2.5 The MAL Header From field of the consumer and the subscription identifier shall form the unique identifier of the subscription; this means that two consumers cannot share subscriptions, as they have different From fields.

3.6.6.2.6 The consumer DEREGISTER Message shall take the list of identifiers of the subscriptions to be cancelled. After, the broker shall completely remove the subscriptions from the list of active subscriptions.

3.6.6.2.7 The update rate shall be an aspect of a provider; the consumer cannot specify it.

NOTES

1 For example, if a consumer initially registers for updates on the values set (‘A’, ‘B’, ‘C’) but at a later date also requires ‘D’, another REGISTER Message is required to be sent with the set (‘A’, ‘B’, ‘C’, ‘D’). The register call replaces the previous subscription list with the new list. The DEREGISTER Message completely cancels the identified lists.

2 Another way of describing this is the REGISTER Message defines an update ‘report’, and the DEREGISTER Message clears that ‘report’ definition.

3 If a single subscription definition lists the same parameter more than once, only one update is sent to the consumer for that list. However, if a consumer defines two concurrent subscription lists that request the same parameters, then the consumer will receive two updates of those common parameters, one for each subscription. If this is an issue, then the consumer still has the option of maintaining a single subscription and splitting the results internally.

3.6.6.3 Broker—Forwarding Published Messages to Consumers

3.6.6.3.1 The Broker forwarding of the published messages is based on two parts, (a) specific fields from the message header and (b) the consumer subscription:

a) The area, service, and operation identifiers of the message header shall form the first part to be matched.

b) The consumer subscription shall form the second part to be matched with the published message. This includes the domain matching, and the specified subscription filters.

3.6.6.4 Broker-Broker—Subscription Filters and Domain Matching

3.6.6.4.1 Overview

This subsection details the rules for subscription filter matching in the PUBLISH-SUBSCRIBE pattern. The subscription is matched on two aspects: the optional filters (which are ANDed together) and the optional domain field. Both of these optional fields are present in the Subscription structure (see 4.6.6).

The following paragraphs detail the rules for filter matching.

3.6.6.4.2 Rules for Filter Matching

3.6.6.4.2.1 If the filters field is set to NULL, then no filtering shall be performed. Therefore any value matches.

3.6.6.4.2.2 If the filters field is not NULL, it shall hold a list of filters composed of a name (type: Identifier) and a set of possible values (type: List< Attribute>) to be matched.

3.6.6.4.2.3 The name field of a filter shall match one of the Subscription Key names specified in the operation template; otherwise the Broker shall return an INTERNAL error to the consumer.

3.6.6.4.2.4 Each filter shall hold a set of possible values to be matched with the Subscription Key value for the corresponding Subscription Key name selected in the filter.

3.6.6.4.2.5 If the list of possible values of a filter is not empty, it shall only match the Subscription Key value that contains one of the values in that list. This includes an empty ‘’ value for text-based data types. The matches shall be case sensitive.

3.6.6.4.2.6 If the list of possible values of a filter is empty, then it shall match the corresponding Subscription Key that contains any value, including NULL.

3.6.6.4.3 Domain Field Matching

3.6.6.4.3.1 If the domain field of the Subscription is set to NULL, then no domain filtering will be performed. Therefore any domain matches.

3.6.6.4.3.2 If the domain field of the Subscription is not set to NULL, then it shall be matched with the domain field of the UpdateHeader of the publish message.

3.6.6.4.3.3 Any of the identifiers of the domain field of the Subscription may be the wildcard character ‘*’.

3.6.6.4.3.4 If one of the identifiers of the domain, in the domain field of the Subscription, is a wildcard, then this domain part shall match any value in the same corresponding part of the domain of the publish message.

3.6.6.4.4 Discussion—Domain Examples

As an example, one can assume that a provider publishes a set of updates with the following domains:

a) spacecraftA;

b) spacecraftA.aocs;

c) spacecraftA.aocs.thrustA;

d) spacecraftA.payload;

e) spacecraftA.payload.cameraA.tempB;

f) spacecraftB;

g) agency.spacecraftA;

h) spacecraftB.payload.cameraA.tempB.

A subscription filter with the domain of 'spacecraftA' would only match the first update.

A subscription filter with the domain of 'spacecraftA.aocs' would only match the second update.

A subscription filter with the domain of 'spacecraftA.payload.*' would match the fourth and fifth updates.

A subscription filter with the domain of '*.payload.cameraA.*' would match the fifth and eighth updates.

A subscription filter with the domain of 'spacecraftA.*' would match updates one to five.

3.6.6.5 Broker—Trimming Keys for the NOTIFY Message

3.6.6.5.1 The consumer subscription shall define the trimming behaviour of the broker for a certain subscription by using the selectedKeys field of the Subscription structure.

3.6.6.5.2 If the selectedKeys field of the Subscription structure is NULL, then the broker shall not trim any of the Subscription Key values.

3.6.6.5.3 If the selectedKeys field of the Subscription structure is not NULL, then the selectedKeys field shall hold the selected list of Subscription Key names.

3.6.6.5.4 If the selectedKeys field of the Subscription structure is not NULL, then the broker shall trim the Subscription Key values list of the NOTIFY message and only hold, in the same order as defined in the selectedKeys field, the Subscription Key values that correspond to the selected list of Subscription Key names.

3.6.6.6 Usage

The PUBLISH-SUBSCRIBE pattern shall provide the ability for asynchronous updates to be sent to registered consumers.

3.6.6.7 Error Handling

3.6.6.7.1 The PUBLISH-SUBSCRIBE pattern can report failure for a consumer in two distinct ways:

a) the register acknowledgement may be replaced with an error message (see section 4) as shown in the first sequence in figure 3-22;

b) a notify may be replaced by an error by the message broker as shown in the second sequence in figure 3-22.

3.6.6.7.2 After an error message is sent, no further messages shall be transmitted to the relevant consumer as part of the pattern.

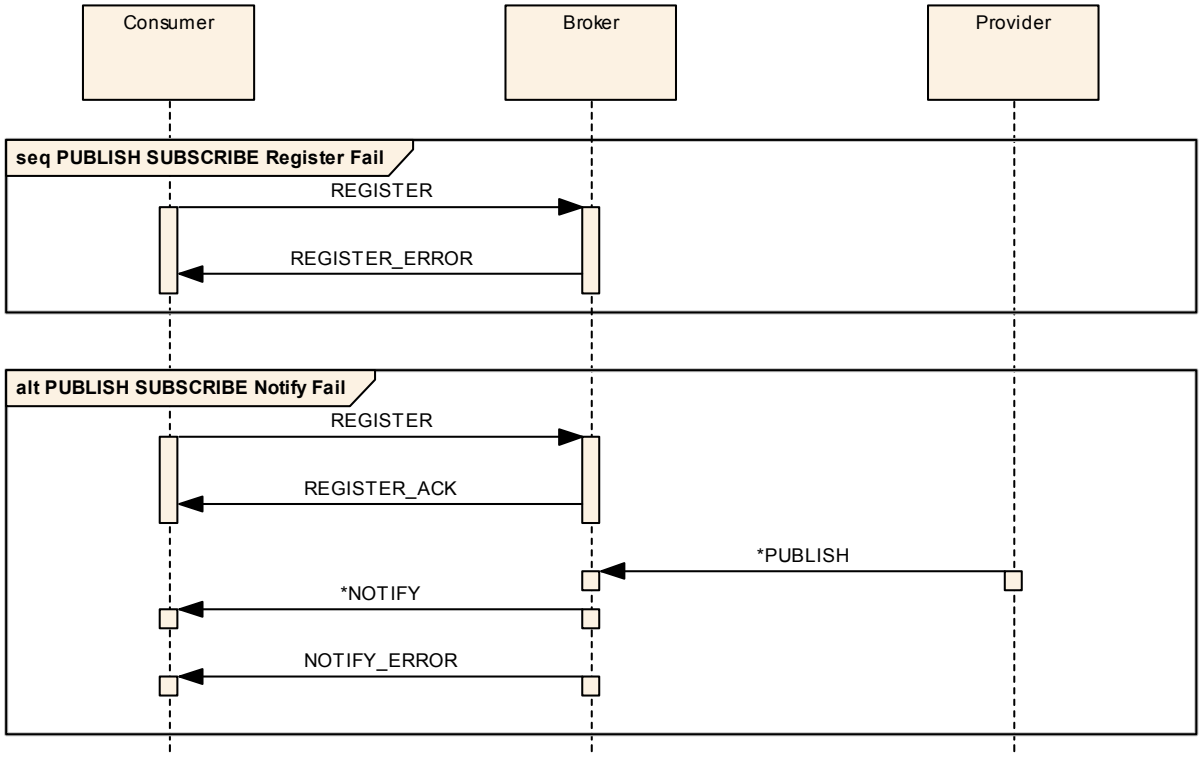

Figure 3-22: PUBLISH-SUBSCRIBE Interaction Pattern Consumer Error Sequence

3.6.6.7.3 For a publisher, the PUBLISH-SUBSCRIBE pattern can report failure in two distinct ways:

a) The publish register acknowledgement may be replaced with an error message (see section 4) as shown in the first sequence in figure 3-23.

b) When an error is detected by the message broker on reception of a PUBLISH Message from a provider, a PUBLISH_ERROR Message shall be sent to the provider as shown in the second sequence in figure 3-23. The error shall be sent asynchronously to the provider.

3.6.6.7.4 After a PUBLISH_REGISTER_ERROR Message is sent, no further messages shall be transmitted to or from the relevant publisher as part of the pattern.

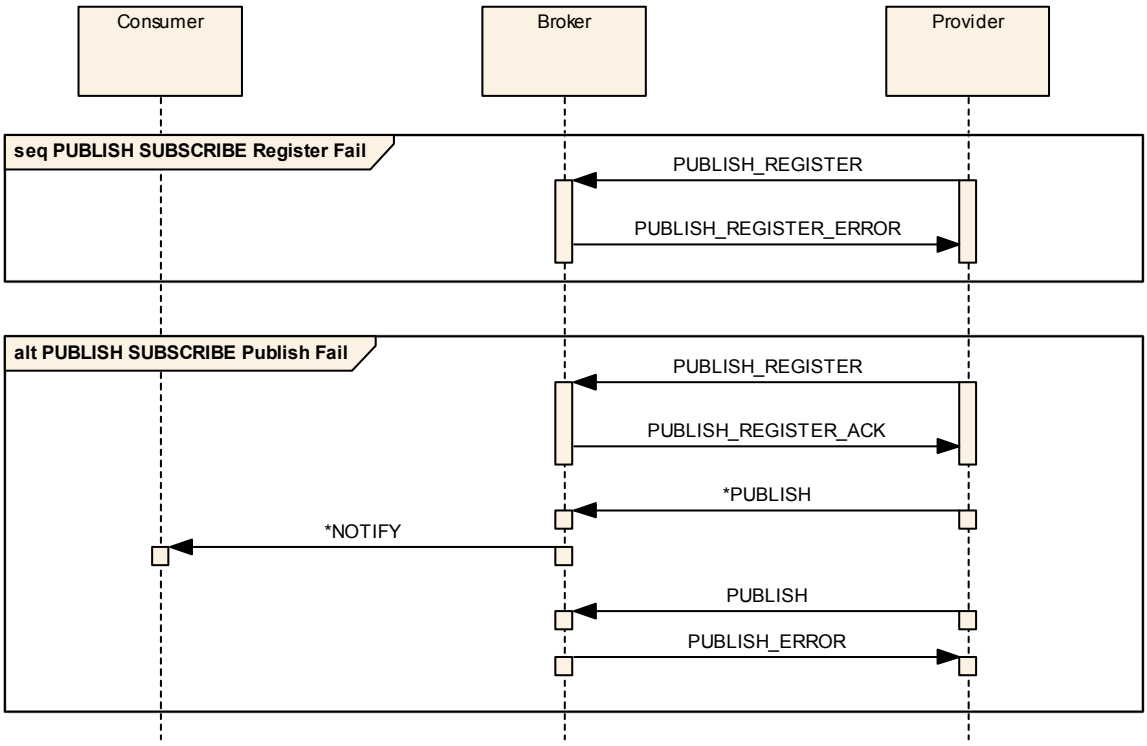

Figure 3-23: PUBLISH-SUBSCRIBE Interaction Pattern Provider Error Sequence

3.6.6.8 Operation Template

3.6.6.8.1 The PUBLISH-SUBSCRIBE pattern shall conform to the PUBLISH-SUBSCRIBE operation templates defined in tables 3-25 through 3-32.

Table 3-24: PUBLISH-SUBSCRIBE Operation Template

Operation Identifier	<<Operation name>>		
Interaction Pattern	PUBLISH-SUBSCRIBE		
Subscription Keys	<<Subscription Key types and names>>		
Pattern Sequence	Message	Nullable	Type Signature
OUT	PUBLISH/NOTIFY	<<Nullability>>	<<Update Signature>>

3.6.6.8.2 The Subscription Keys row shall present the list of default Subscription Keys for the defined operation. It shall include the name and its respective data type for each Subscription Key.

3.6.6.8.3 The specified types of the Subscription Keys shall always be Concrete Attribute types (e.g., String, Identifier, Long, etc.).

3.6.6.8.4 The default set of Subscription Keys in the operation template shall always be implemented and supported as part of the operation.

3.6.6.8.5 Concrete implementations of a certain PUBLISH-SUBSCRIBE operation may add additional custom Subscription Keys. These additional custom Subscription Keys are complementary to the default set and shall not replace the default set.

3.6.6.8.6 If custom Subscription Keys are to be used by an implementation, then an out-of-band agreement shall be defined for their usage.

3.6.6.8.7 The PUBLISH/NOTIFY Message shall follow the same rules for message bodies (as defined in 3.5.3), in which the Message Body may be composed of several types.

NOTE – Subsection 3.6.6.12 provides an example of this for this Interaction Pattern.

3.6.6.8.8 A single update shall require a value for each type listed in the operation template in the case in which it is defined using multiple types.

NOTE – Subsection 3.6.6.12 provides an example of this for this Interaction Pattern.

3.6.6.8.9 The update signature shall be the same as it is passed in the provider-to-broker PUBLISH and broker-to-consumer NOTIFY Messages.

NOTE – The contents of these messages are fixed and therefore not present in the operation template for PUBLISH-SUBSCRIBE. The PUBLISH-SUBSCRIBE pattern is actually made up of consumer and provider REGISTER/DEREGISTER, PUBLISH, and NOTIFY Messages, the templates of which are given below. The message directions (IN/OUT) are all from the point of view of the broker, as it is involved in all operations:

Table 3-25: PUBLISH-SUBSCRIBE Register Operation Template

Operation Identifier	register<< PUBLISH-SUBSCRIBE Operation Name>>		
Interaction Pattern	SUBMIT		
Pattern Sequence	Message	Nullable	Type Signature
IN	REGISTER	No	Subscription subscription

3.6.6.8.10 The consumer register method takes a subscription which is composed of an identifier, a domain, and a list of filters. The identifier shall be used by the broker to uniquely identify the subscription when combined with the From field of the register message; this allows a consumer to set up several different concurrent subscriptions.

3.6.6.8.11 Each subscription shall contain a list of filters which contain an identifier key name and a list of values to be matched.

Table 3-26: PUBLISH-SUBSCRIBE Publish Register Operation Template

Operation Identifier	publishRegister<< PUBLISH-SUBSCRIBE Operation Name>>		
Interaction Pattern	SUBMIT		
Pattern Sequence	Message	Nullable	Type Signature
IN	PUBLISH_REGISTER	No No	List<Identifier> names List<AttributeType> types

3.6.6.8.12 The PUBLISH_REGISTER takes two arguments, the first a list of Subscription Key names, and the second a list of their respective Attribute types.

3.6.6.8.13 The PUBLISH_REGISTER message shall register in the Broker, the provider from which updates will be performed.

Table 3-27: PUBLISH-SUBSCRIBE Publish Operation Template

Operation Identifier	publish<< PUBLISH-SUBSCRIBE Operation Name>>		
Interaction Pattern	SEND		
Pattern Sequence	Message	Nullable	Type Signature
IN	PUBLISH	No <<Nullability>>	UpdateHeader header <<Update Signature>>

3.6.6.8.14 The provider-to-broker PUBLISH Message shall match the PUBLISH_REGISTER on the area identifier, service identifier, area version identifier, and operation identifier.

3.6.6.8.15 A PUBLISH Message Body shall contain an UpdateHeader followed by the Update Signature defined in the top-level PUBSUB template. For example, a PUBSUB operation defined using the types of [Boolean, Octet, MyComposite] will result in a PUBLISH Message with the body containing [UpdateHeader, Boolean, Octet, MyComposite].

3.6.6.8.16 For each update being published, there shall be an UpdateHeader and the corresponding Update Signature.

3.6.6.8.17 The UpdateHeader shall optionally contain the source of the update (implementation-specific), the domain, and the update key values.

3.6.6.8.18 The key values in the UpdateHeader structure shall be ordered as defined by the operation so that the corresponding key names can be matched with the entries in the list.

3.6.6.8.19 If a provider has implemented custom Subscription Keys, then the corresponding Subscription Key values shall be on the list after the default set of Subscription Key values.

Table 3-28: PUBLISH-SUBSCRIBE Publish Error Operation Template

Operation Identifier	publishError<< PUBLISH-SUBSCRIBE Operation Name>>		
Interaction Pattern	SEND		
Pattern Sequence	Message	Nullable	Type Signature
OUT	PUBLISH_ERROR	No	UInteger errorNumber

3.6.6.8.20 If a provider publishes an update with more or less key values than expected, then the broker shall return an UNKNOWN error in a PUBLISH_ERROR Message.

Table 3-29: PUBLISH-SUBSCRIBE Notify Operation Template

Operation Identifier	notify<< PUBLISH-SUBSCRIBE Operation Name>>		
Interaction Pattern	SEND		
Pattern Sequence	Message	Nullable	Type Signature
OUT	NOTIFY	No No <<Nullability>>	Identifier subscriptionId UpdateHeader updateHeader <<Update Signature>>

3.6.6.8.21 The broker-to-consumer NOTIFY Message Body shall contain a single identifier, an UpdateHeader followed by the Update Signature defined in the top-level PUBSUB template, which holds the update set for a single subscription. For example, a PUBSUB operation defined using the types of [Boolean, Octet, MyComposite] will result in a NOTIFY Message with the body containing [Identifier, UpdateHeader, Boolean, Octet, MyComposite].

3.6.6.8.22 If the consumer subscription did not enable trimming, then the key values in the UpdateHeader message shall be ordered as defined by the operation so that the corresponding key names can be matched with the entries in the list.

3.6.6.8.23 If the consumer subscription did enable trimming, then the key values in the UpdateHeader message shall be ordered as defined according to 3.6.6.5.

3.6.6.8.24 The single identifier in the NOTIFY Message shall contain the subscription identifier for the subscription being notified.

3.6.6.8.25 In the NOTIFY Message there shall be the Update Signatures for each type defined in the top level PUBSUB template.

3.6.6.8.26 If all entries in the UpdateHeader list are NULL, then the whole list shall be replaced with a NULL in order to increase efficiency.

Table 3-30: PUBLISH-SUBSCRIBE Notify Error Operation Template

Operation Identifier	notifyError<< PUBLISH-SUBSCRIBE Operation Name>>		
Interaction Pattern	SEND		
Pattern Sequence	Message	Nullable	Type Signature
OUT	NOTIFY_ERROR	No Yes	UInteger errorNumber Element extraInformation

3.6.6.8.27 If the broker wants to shut down, then the broker shall return a SHUTDOWN error in a NOTIFY_ERROR Message and an Extra Information message on the cause.

3.6.6.8.28 If the broker has an internal error, then the broker shall return an INTERNAL error in a NOTIFY_ERROR Message and an Extra Information message on the cause.

Table 3-31: PUBLISH-SUBSCRIBE Deregister Operation Template

Operation Identifier	deregister<< PUBLISH-SUBSCRIBE Method Name>>		
Interaction Pattern	SUBMIT		
Pattern Sequence	Message	Nullable	Type Signature
IN	DEREGISTER	No	List<Identifier> subscriptionIds

3.6.6.8.29 The consumer DEREGISTER Message shall take a list of identifiers of the active subscription lists to cancel.

Table 3-32: PUBLISH-SUBSCRIBE Publish Deregister Operation Template

Operation Identifier	publishDeregister<< PUBLISH-SUBSCRIBE Method Name>>		
Interaction Pattern	SUBMIT		
Pattern Sequence	Message	Nullable	Type Signature
IN	PUBLISH_DEREGISTER		

3.6.6.8.30 The provider DEREGISTER Message takes no arguments. It shall completely remove the provider from the set of providers from which updates are permitted.

3.6.6.8.31 In all cases, for consumer/provider register and deregister, the acknowledge message shall only be sent when the registration/deregistration has completed and is a confirmation that the operation has succeeded.

3.6.6.9 Primitives

Table 3-33: PUBLISH-SUBSCRIBE Primitive List

Primitive
REGISTER
REGISTER_ACK
REGISTER_ERROR
PUBLISH_REGISTER
PUBLISH_REGISTER_ACK
PUBLISH_REGISTER_ERROR
PUBLISH
PUBLISH_ERROR
NOTIFY
NOTIFY_ERROR
DEREGISTER
DEREGISTER_ACK
PUBLISH_DEREGISTER
PUBLISH_DEREGISTER_ACK

3.6.6.10 State Charts

3.6.6.10.1 Overview

Several state charts are defined:

- one state chart on the consumer side related to the interaction between the consumer and the broker;
- one state chart on the broker side related to the interaction between the consumer and the broker;
- one state chart on the provider side related to the interaction between the provider and the broker;
- another state chart on the broker side related to the interaction between the provider and the broker.

3.6.6.10.2 Consumer Side

3.6.6.10.2.1 Figure 3-24 is the consumer side state chart for the PUBLISH-SUBSCRIBE pattern.

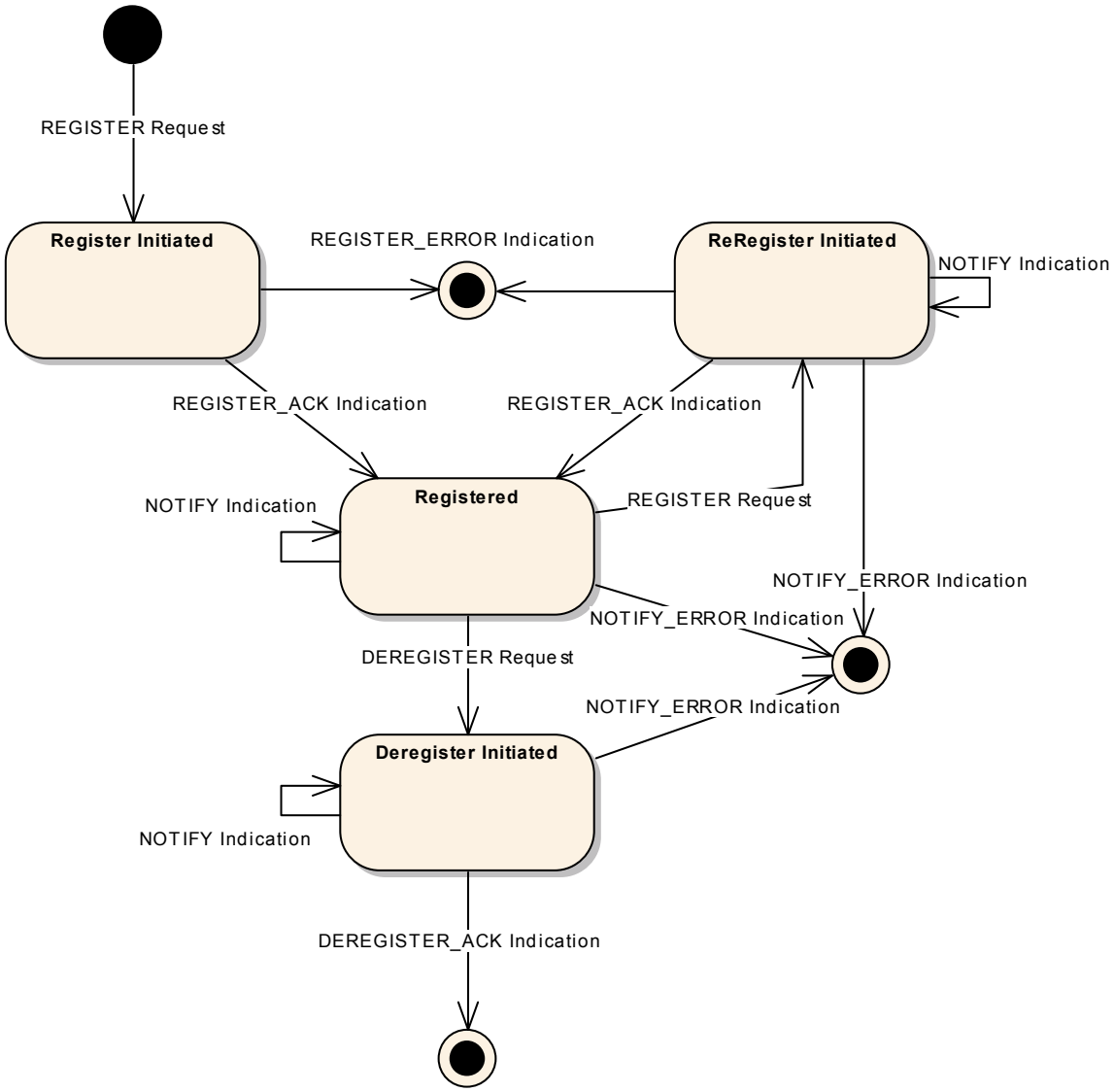

Figure 3-24: PUBLISH-SUBSCRIBE Consumer State Chart

3.6.6.10.2.2 The PUBLISH-SUBSCRIBE consumer state chart applies to a single subscription for a single consumer. The state chart shall be replicated for each consumer and for each subscription.

3.6.6.10.2.3 The Transaction Id of the initial REGISTER Message shall be used to identify NOTIFY messages that relate to one PUBLISH-SUBSCRIBE interaction to avoid race conditions.

3.6.6.10.2.4 The initial transition shall be triggered by the primitive REGISTER Request and shall lead to the Register Initiated State.

3.6.6.10.2.5 There are two possible transitions from the Register Initiated State:

a) the first is the indication of an acknowledgement, REGISTER_ACK Indication, and shall lead to the Registered State;

b) the second is the indication of an error, REGISTER_ERROR Indication, and shall lead to the final state.

3.6.6.10.2.6 There are four possible transitions from the Registered State:

a) the first is the indication of a notification, NOTIFY Indication, and shall lead back to the Registered State;

b) the second is the indication of an error, NOTIFY_ERROR Indication, and shall lead to the final state;

c) the third shall be triggered by the primitive DEREGISTER Request and shall lead to the Deregister Initiated State;

d) the fourth shall be triggered by the primitive REGISTER Request and shall lead to the ReRegister Initiated State.

3.6.6.10.2.7 There are four possible transitions from the ReRegistered Initiated State:

a) The first is the indication of a notification, NOTIFY Indication, and shall lead back to the ReRegistered Initiated State. In this case the NOTIFY Message will have been sent before the REGISTER Message was received by the broker and therefore forms part of the previous subscription. It shall, however, be passed to the consumer application as normal.

b) The second is the indication of a registration error, REGISTER_ERROR Indication, and shall lead to the final state.

c) The third is the indication of a notify error, NOTIFY_ERROR Indication, and shall lead to the final state. In this case, the NOTIFY_ERROR Message will have been sent before the REGISTER Message was received by the broker and therefore forms part of the previous subscription. It shall, however, be passed to the consumer application as normal. The Transaction Id shall be used by the broker to ensure that the reception of the REGISTER Message after the NOTIFY_ERROR Message was sent does not cause the restart of the subscription.

d) The fourth is the indication of an acknowledgement, REGISTER_ACK Indication, and shall lead to the Registered State.

3.6.6.10.2.8 There are three possible transitions from the Deregister Initiated State:

a) the first is the indication of a notification, NOTIFY Indication, and shall lead back to the Deregister Initiated State;

b) the second is the indication of an error, NOTIFY_ERROR Indication, and shall lead to the final state;

c) the third is the indication of an acknowledgement, DEREGISTER_ACK Indication, and shall lead to the final state.

3.6.6.10.3 Broker Side (Related to the Consumer)

3.6.6.10.3.1 Figure 3-25 is the broker-to-consumer state chart for the PUBLISH-SUBSCRIBE pattern.

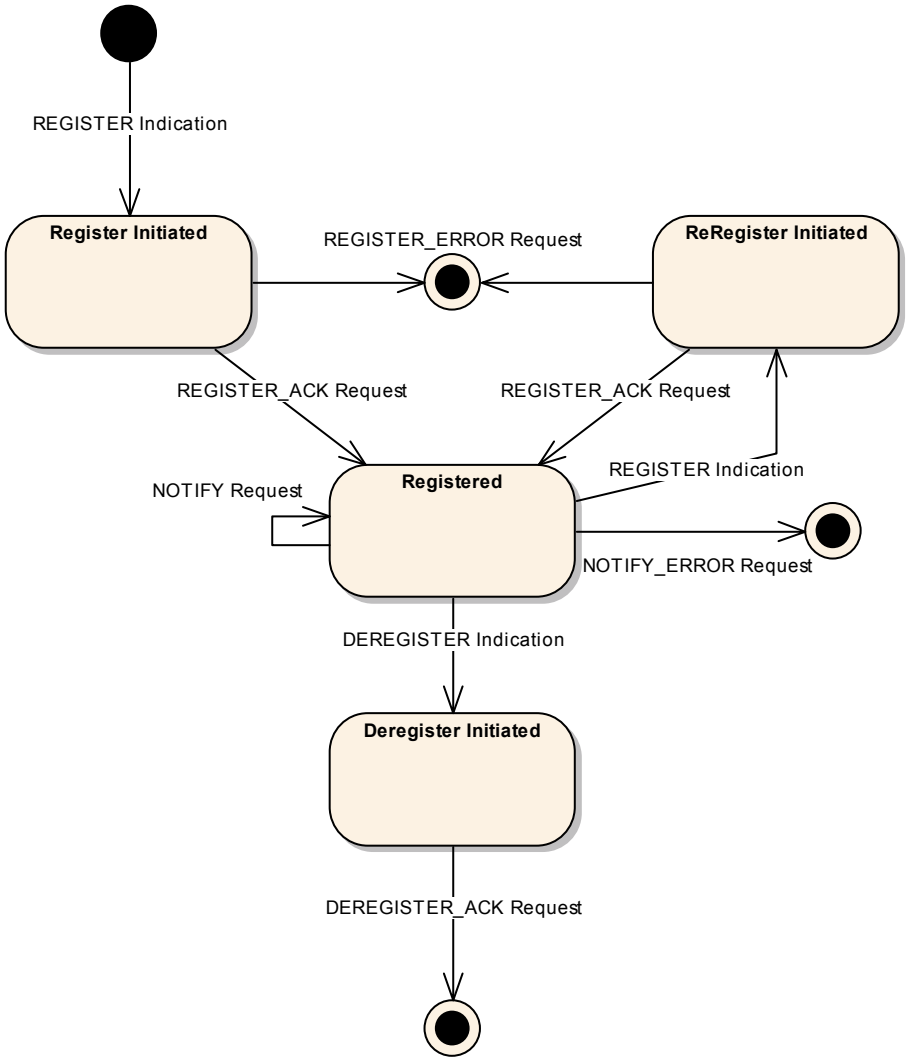

Figure 3-25: PUBLISH-SUBSCRIBE Broker to Consumer State Chart

3.6.6.10.3.2 The PUBLISH-SUBSCRIBE broker-to-consumer state chart applies to a single subscription for a single consumer. The state chart shall be replicated for each consumer and for each subscription.

3.6.6.10.3.3 The initial transition is triggered by the primitive REGISTER Indication and shall lead to the Register Initiated State.

3.6.6.10.3.4 There are two possible transitions from the Register Initiated State:

a) the first is the triggering of an acknowledgement, REGISTER_ACK Request, and shall lead to the Registered State;

b) the second is the triggering of an error, REGISTER_ERROR Request, and shall lead to the final state.

3.6.6.10.3.5 There are four possible transitions from the Registered State:

a) the first is the triggering of a notification, NOTIFY Request, and shall lead back to the Registered State;

b) the second is the triggering of an error, NOTIFY_ERROR Request, and shall lead to the final state;

c) the third shall be triggered by the primitive DEREGISTER Indication and shall lead to the Deregister Initiated State;

d) the fourth shall be triggered by the primitive REGISTER Indication and shall lead to the ReRegister Initiated State.

3.6.6.10.3.6 There are two possible transitions from the ReRegister Initiated State:

a) the first is the triggering of an acknowledgement, REGISTER_ACK Request, and shall lead to the Registered State;

b) the second is the triggering of an error, REGISTER_ERROR Request, and shall lead to the final state.

3.6.6.10.3.7 There is only one possible transition from the Deregister Initiated State; it is the triggering of an acknowledgement, DEREGISTER_ACK Request, and shall lead to the final state.

3.6.6.10.4 Provider Side

3.6.6.10.4.1 Figure 3-26 is the provider side state chart for the PUBLISH-SUBSCRIBE pattern.

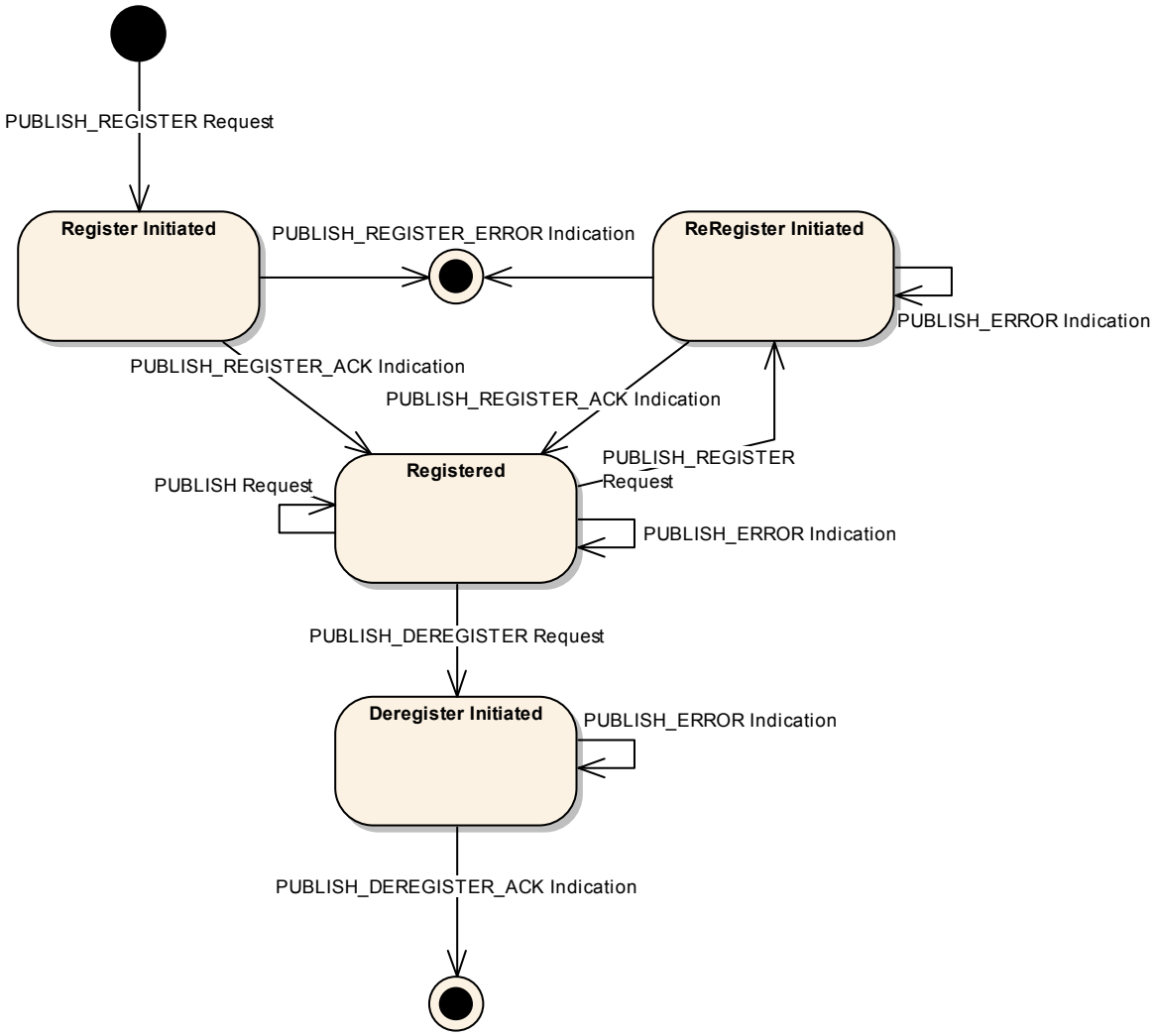

Figure 3-26: PUBLISH-SUBSCRIBE Provider State Chart

3.6.6.10.4.2 The PUBLISH-SUBSCRIBE provider state chart applies to a single provider. The state chart shall be replicated for each provider.

3.6.6.10.4.3 The initial transition is triggered by the primitive PUBLISH_REGISTER Request and shall lead to the Register Initiated State.

3.6.6.10.4.4 There are two possible transitions from the Register Initiated State:

a) the first is the indication of an acknowledgement, PUBLISH_REGISTER_ACK Indication, and shall lead to the Registered State;

b) the second is the indication of an error, PUBLISH_REGISTER_ERROR Indication, and shall lead to the final state.

3.6.6.10.4.5 There are four possible transitions from the Registered State:

a) the first is the triggering of a publish, PUBLISH Request, and shall lead back to the Registered State;

b) the second is the indication of an error, PUBLISH_ERROR Indication, and shall lead back to the Registered State;

c) the third shall be triggered by the primitive PUBLISH_REGISTER Request and shall lead to the ReRegister Initiated State;

d) the fourth shall be triggered by the primitive PUBLISH_DEREGISTER Request and shall lead to the Deregister Initiated State.

3.6.6.10.4.6 There are three possible transitions from the ReRegister Initiated State:

a) The first is the indication of an acknowledgement, PUBLISH_REGISTER_ACK Indication, and shall lead to the Registered State.

b) The second is the indication of a registration error, PUBLISH_REGISTER_ERROR Indication, and shall lead to the final state.

c) The third is the indication of a publish error, PUBLISH_ERROR Indication, and shall lead back to the Re-Registered State. In this case, the PUBLISH_ERROR Message will have been sent before the PUBLISH_REGISTER Message was received by the broker and therefore forms part of the previous subscription.

3.6.6.10.4.7 There are two possible transitions from the Deregister Initiated State:

a) the first is the indication of an acknowledgement, PUBLISH_DEREGISTER_ACK Indication, and shall lead to the final state;

b) the second is the indication of a publish error, PUBLISH_ERROR Indication, and shall lead back to the Deregister Initiated State.

3.6.6.10.5 Broker Side (Related to the Provider)

3.6.6.10.5.1 Figure 3-27 is the broker-to-provider state chart for the PUBLISH-SUBSCRIBE pattern.

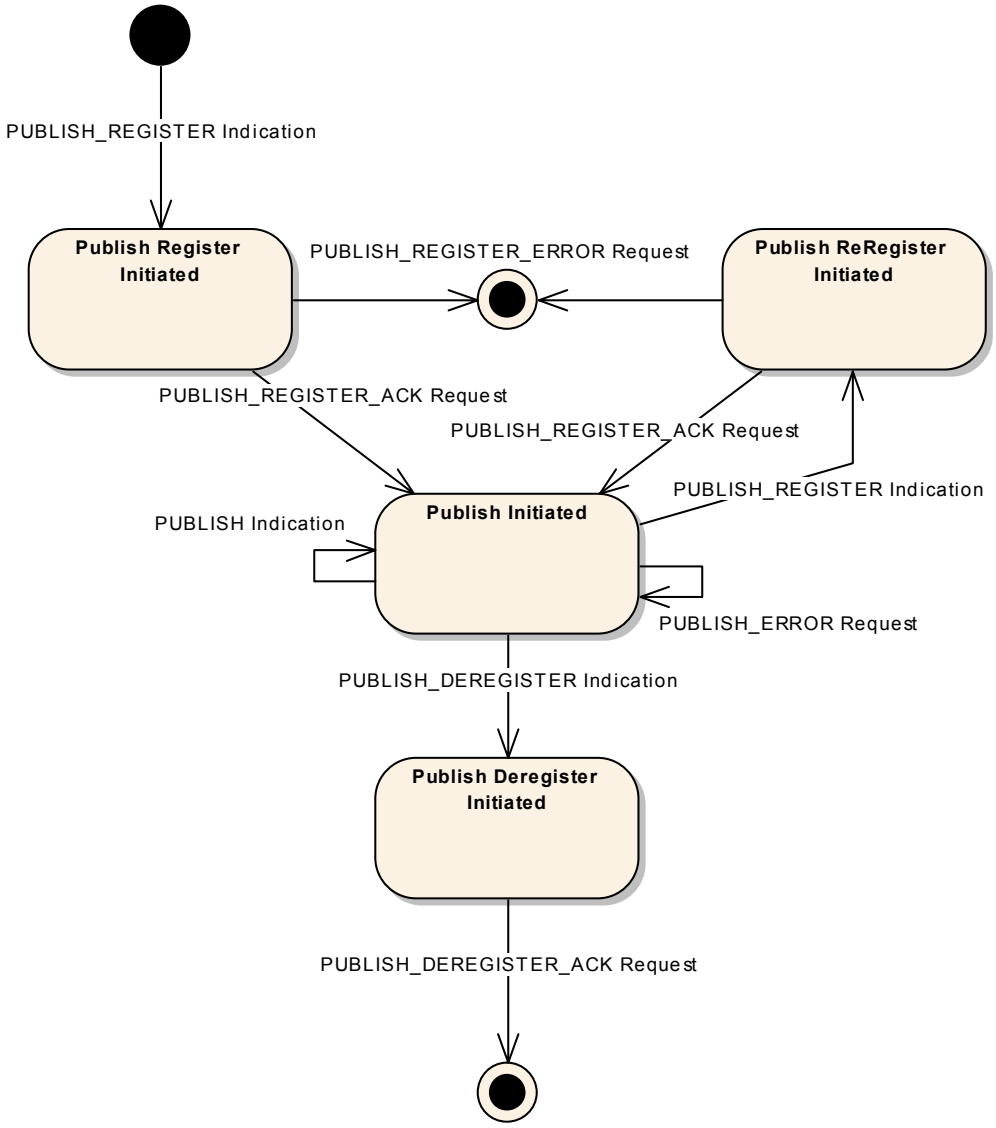

Figure 3-27: PUBLISH-SUBSCRIBE Broker to Provider State Chart

3.6.6.10.5.2 The PUBLISH-SUBSCRIBE broker-to-provider state chart applies to a single provider. The state chart shall be replicated for each provider.

3.6.6.10.5.3 The initial transition is triggered by the primitive PUBLISH_REGISTER Indication and shall lead to the Publish Register Initiated State.

3.6.6.10.5.4 There are two possible transitions from the Publish Register Initiated State:

a) the first is the triggering of an acknowledgement, PUBLISH_REGISTER_ACK Request, and shall lead to the Publish Initiated State;

b) the second is the triggering of an error, PUBLISH_REGISTER_ERROR Request, and shall lead to the final state.

3.6.6.10.5.5 There are four possible transitions from the Publish Initiated State:

a) the first is the indication of a publish, PUBLISH Indication, and shall lead back to the Publish Initiated State;

b) the second is the triggering of an error, PUBLISH_ERROR Request, and shall lead back to the Publish Initiated State;

c) the third shall be triggered by the primitive PUBLISH_DEREGISTER Indication and shall lead to the Publish Deregister Initiated State;

d) the fourth shall be triggered by the primitive PUBLISH_REGISTER Indication and shall lead to the Publish ReRegister Initiated State.

3.6.6.10.5.6 There are two possible transitions from the Publish ReRegister Initiated State:

a) the first is the triggering of an acknowledgement, PUBLISH_REGISTER_ACK Request, and shall lead to the Publish Initiated State;

b) the second is the triggering of an error, PUBLISH_REGISTER_ERROR Request, and shall lead to the final state.

3.6.6.10.5.7 There is only one possible transition from the Publish Deregister Initiated State; it is the triggering of an acknowledgement, PUBLISH_DEREGISTER_ACK Request, and shall lead to the final state.

3.6.6.11 Requests and Indications

3.6.6.11.1 REGISTER

3.6.6.11.1.1 Function

3.6.6.11.1.1.1 The REGISTER Request primitive shall be used by the consumer application to initiate a PUBSUB interaction with a subscription or to update an existing subscription for the specified subscription identifier.

3.6.6.11.1.1.2 The REGISTER Indication primitive shall be used by the broker MAL to deliver a REGISTER Message to a broker and initiate a PUBSUB interaction or to update an existing subscription for the specified subscription identifier.

3.6.6.11.1.2 Semantics

REGISTER Request and REGISTER Indication shall provide parameters as follows:

(REGISTER Message Header, Subscription)

3.6.6.11.1.3 When Generated

3.6.6.11.1.3.1 A REGISTER Request may be used by the consumer application at any time to start a new subscription or when the consumer MAL is in the Registered State to update an existing subscription.

3.6.6.11.1.3.2 A REGISTER Indication shall be generated by the broker MAL for a new subscription or when the broker MAL is in the Registered State for existing subscriptions for that consumer upon reception of a REGISTER Message, once checked via the Access Control interface, from a consumer.

3.6.6.11.1.4 Effect on Reception

3.6.6.11.1.4.1 Reception of a REGISTER Request shall, once checked via the Access Control interface, cause the consumer MAL to initiate a PUBSUB interaction by transmitting a REGISTER Message to the broker for a new subscription.

3.6.6.11.1.4.2 The consumer MAL shall enter the Register Initiated State in this case.

3.6.6.11.1.4.3 If the consumer MAL is in the Registered State, reception of a REGISTER Request shall, once checked via the Access Control interface, cause the consumer MAL to update the subscription by transmitting a REGISTER Message to the broker.

3.6.6.11.1.4.4 The consumer MAL shall enter the ReRegister Initiated State in this case.

3.6.6.11.1.4.5 On reception of a REGISTER Message by the broker MAL, once the message has been checked via the Access Control interface, the broker MAL shall enter the Register Initiated State if this is the first message in a new PUBSUB interaction, or enter the ReRegister Initiated State if currently in the Registered State and then pass a REGISTER Indication to the broker.

3.6.6.11.1.4.6 The broker shall store the subscription of the consumer or update an existing subscription from the same consumer if the subscription identifier is already used by that consumer.

3.6.6.11.1.4.7 The broker shall start processing the subscription at this point.

3.6.6.11.1.5 Message Header

3.6.6.11.1.5.1 For the REGISTER Message, the Message Header fields shall be as defined in 3.5 except for the fields in table 3-34.

3.6.6.11.1.5.2 The contents of the Subscription structure, as specified in the operation template, shall immediately follow the Message Header.

Table 3-34: REGISTER Message Header Fields

Field	Value
From	Consumer identifier
Authentication Id	Consumer Authentication Identifier
To	Broker identifier
Interaction Type	PUBSUB
Interaction Stage	1
Transaction Id	Transaction Id from consumer MAL

3.6.6.11.2 REGISTER_ACK

3.6.6.11.2.1 Function

3.6.6.11.2.1.1 The REGISTER_ACK Request primitive shall be used by the broker to acknowledge a PUBSUB interaction subscription.

3.6.6.11.2.1.2 The REGISTER_ACK Indication primitive shall be used by the consumer MAL to deliver a REGISTER_ACK Message to a consumer.

3.6.6.11.2.2 Semantics

REGISTER_ACK Request and REGISTER_ACK Indication shall provide parameters as follows:

(REGISTER_ACK Message Header)

3.6.6.11.2.3 When Generated

3.6.6.11.2.3.1 A REGISTER_ACK Request shall be used by the broker with the broker MAL in either the Register Initiated State or ReRegister Initiated State to acknowledge the successful registration by a consumer in a PUBSUB interaction.

3.6.6.11.2.3.2 A REGISTER_ACK Indication shall be generated by the consumer MAL in either the Register Initiated State or ReRegister Initiated State upon reception of a REGISTER_ACK Message, once checked via the Access Control interface, from a broker.

3.6.6.11.2.4 Effect on Reception

3.6.6.11.2.4.1 Reception of a REGISTER_ACK Request shall, once checked via the Access Control interface, cause the broker MAL to transmit the supplied acknowledgement as a REGISTER_ACK Message to the consumer.

3.6.6.11.2.4.2 A broker MAL in the Register Initiated State or ReRegister Initiated State shall enter the Registered State.

3.6.6.11.2.4.3 On reception of a REGISTER_ACK Message by the consumer MAL, once the message has been checked via the Access Control interface, the consumer MAL shall enter the Registered State and pass a REGISTER_ACK Indication to the consumer application.

3.6.6.11.2.5 Message Header

For the REGISTER_ACK Message, the Message Header fields shall be as defined in 3.5 except for the fields in table 3-35.

Table 3-35: REGISTER_ACK Message Header Fields

Field	Value
From	Broker identifier
Authentication Id	Broker Authentication Identifier
To	Consumer identifier
Interaction Type	PUBSUB
Interaction Stage	2
Transaction Id	Transaction Id from REGISTER Message

3.6.6.11.3 REGISTER_ERROR

3.6.6.11.3.1 Function

3.6.6.11.3.1.1 The REGISTER_ERROR Request primitive shall be used by the broker to reject a subscription request and end a PUBSUB interaction with an error.

3.6.6.11.3.1.2 The REGISTER_ERROR Indication primitive shall be used by the consumer MAL to deliver a REGISTER_ERROR Message to a consumer.

3.6.6.11.3.2 Semantics

REGISTER_ERROR Request and REGISTER_ERROR Indication shall provide parameters as follows:

(REGISTER_ERROR Message Header, Error Number, Extra Information)

3.6.6.11.3.3 When Generated

3.6.6.11.3.3.1 A REGISTER_ERROR Request shall be used by the broker with the broker MAL in either the Register Initiated State or ReRegister Initiated State to reject a REGISTER Request.

3.6.6.11.3.3.2 The broker may reject the register attempt for one of the following reasons:

a) More subscriptions than the broker can support shall cause a TOO_MANY error to be sent. The Extra Information field of the error contains an integer which provides the maximum number of subscriptions supported.

b) Shutdown: the broker is shutting down. A SHUTDOWN error message shall be returned.

c) Internal error: the broker has experienced an internal error. An INTERNAL error message shall be returned.

d) UNKNOWN shall not be generated for subscriptions that identify keys that have not been currently registered by a publisher. This is because the consumer cannot know what entities are currently provided and which providers are currently registered.

3.6.6.11.3.3.3 A REGISTER_ERROR Indication shall be generated by the consumer MAL in either the Register Initiated State or ReRegister Initiated State upon one of two events:

a) the reception of a REGISTER_ERROR Message, once checked via the Access Control interface, from a broker;

b) an error raised by the local communication layer.

3.6.6.11.3.4 Effect on Reception

3.6.6.11.3.4.1 Reception of a REGISTER_ERROR Request shall, once checked via the Access Control interface, cause the broker MAL to transmit a REGISTER_ERROR Message to the consumer.

3.6.6.11.3.4.2 The pattern shall end at this point for the broker.

3.6.6.11.3.4.3 On reception of a REGISTER_ERROR Message by the consumer MAL, once the message has been checked via the Access Control interface, the consumer MAL shall end the interaction by passing the error to the consumer application.

3.6.6.11.3.5 Message Header

3.6.6.11.3.5.1 For the REGISTER_ERROR Message, the Message Header fields shall be the same as for the REGISTER_ACK Message except for the 'Is Error Message' field, which is set to TRUE.

3.6.6.11.3.5.2 The Error Information shall immediately follow the Message Header.

3.6.6.11.4 PUBLISH_REGISTER

3.6.6.11.4.1 Function

3.6.6.11.4.1.1 The PUBLISH_REGISTER Request primitive shall be used by the provider application to initiate a PUBSUB interaction with the list of Subscription Key names and the respective types being published or to update an existing list.

3.6.6.11.4.1.2 The PUBLISH_REGISTER Indication primitive shall be used by the broker MAL to deliver a PUBLISH_REGISTER Message to a broker and initiate a PUBSUB interaction or to update an existing publish list if one exists for the specified provider.

3.6.6.11.4.2 Semantics

PUBLISH_REGISTER Request and PUBLISH_REGISTER Indication shall provide parameters as follows:

(PUBLISH_REGISTER Message Header, List<Identifier>, List<AttributeType>)

3.6.6.11.4.3 When Generated

3.6.6.11.4.3.1 A PUBLISH_REGISTER Request may be generated by the provider application at any time to start a publishing session or when the provider MAL is already in the Registered State to update an existing Subscription Keys list.

3.6.6.11.4.3.2 A PUBLISH_REGISTER Indication shall be generated by the broker MAL if a new publisher is detected or in the Publish Registered State for the provider upon reception of a PUBLISH_REGISTER Message, once checked via the Access Control interface, from a provider.

3.6.6.11.4.4 Effect on Reception

3.6.6.11.4.4.1 Reception of a PUBLISH_REGISTER Request shall, once checked via the Access Control interface, cause the provider MAL to initiate a PUBSUB interaction by transmitting a PUBLISH_REGISTER Message to the broker.

3.6.6.11.4.4.2 The provider MAL shall enter the Register Initiated State in this case.

3.6.6.11.4.4.3 If the provider MAL is already in the Registered State, reception of a PUBLISH_REGISTER Request shall, once checked via the Access Control interface, cause the provider MAL to transmit a PUBLISH_REGISTER Message to the broker.

3.6.6.11.4.4.4 The provider MAL shall enter the ReRegister Initiated State in this case.

3.6.6.11.4.4.5 On reception of a PUBLISH_REGISTER Message by the broker MAL, once the message has been checked via the Access Control interface, the broker MAL shall enter the Publish Register Initiated State if this is the first message in a new PUBSUB interaction with the provider, or enter the Publish ReRegister Initiated State if currently in the Publish Registered State with that provider, and then pass a PUBLISH_REGISTER Indication to the broker.

3.6.6.11.4.5 Message Header

For the PUBLISH_REGISTER Message, the Message Header fields shall be as defined in 3.5 except for the fields in table 3-36.

Table 3-36: PUBLISH_REGISTER Message Header Fields

Field	Value
From	Provider identifier
Authentication Id	Provider Authentication Identifier
To	Broker identifier
Interaction Type	PUBSUB
Interaction Stage	3
Transaction Id	Transaction Id from provider MAL

3.6.6.11.5 PUBLISH_REGISTER_ACK

3.6.6.11.5.1 Function

3.6.6.11.5.1.1 The PUBLISH_REGISTER_ACK Request primitive shall be used by the broker to acknowledge a PUBSUB interaction publish list from a provider.

3.6.6.11.5.1.2 The PUBLISH_REGISTER_ACK Indication primitive shall be used by the provider MAL to deliver a PUBLISH_REGISTER_ACK Message to a provider.

3.6.6.11.5.2 Semantics

PUBLISH_REGISTER_ACK Request and PUBLISH_REGISTER_ACK Indication shall provide parameters as follows:

(PUBLISH_REGISTER Message Header)

3.6.6.11.5.3 When Generated

3.6.6.11.5.3.1 A PUBLISH_REGISTER_ACK Request shall be used by the broker with the broker MAL in either the Publish Register Initiated State or Publish ReRegister Initiated State to acknowledge the successful registration by a provider in a PUBSUB interaction.

3.6.6.11.5.3.2 A PUBLISH_REGISTER_ACK Indication shall be generated by the provider MAL in either the Register Initiated State or ReRegister Initiated State upon reception of a PUBLISH_REGISTER_ACK Message, once checked via the Access Control interface, from a broker.

3.6.6.11.5.4 Effect on Reception

3.6.6.11.5.4.1 Reception of a PUBLISH_REGISTER_ACK Request shall, once checked via the Access Control interface, cause the broker MAL to transmit the supplied acknowledgement as a PUBLISH_REGISTER_ACK Message to the provider.

3.6.6.11.5.4.2 A broker MAL in the Publish Register Initiated State or Publish ReRegister Initiated State shall then enter the Registered State.

3.6.6.11.5.4.3 On reception of a PUBLISH_REGISTER_ACK Message by the provider MAL, once the message has been checked via the Access Control interface, the provider MAL shall enter the Registered State and pass a PUBLISH_REGISTER_ACK Indication to the provider application.

3.6.6.11.5.5 Message Header

For the PUBLISH_REGISTER_ACK Message, the Message Header fields shall be as defined in 3.5 except for the fields in table 3-37.

Table 3-37: PUBLISH_REGISTER_ACK Message Header Fields

Field	Value
From	Broker identifier
Authentication Id	Broker Authentication Identifier
To	Provider identifier
Interaction Type	PUBSUB
Interaction Stage	4
Transaction Id	Transaction Id from PUBLISH_REGISTER Message

3.6.6.11.6 PUBLISH_REGISTER_ERROR

3.6.6.11.6.1 Function

3.6.6.11.6.1.1 The PUBLISH_REGISTER_ERROR Request primitive shall be used by the broker to reject a PUBLISH REGISTRATION Request from a provider and end a PUBSUB interaction with an error.

3.6.6.11.6.1.2 The PUBLISH_REGISTER_ERROR Indication primitive shall be used by the provider MAL to deliver a PUBLISH_REGISTER_ERROR Message to a provider.

3.6.6.11.6.2 Semantics

PUBLISH_REGISTER_ERROR Request and PUBLISH_REGISTER_ERROR Indication shall provide parameters as follows:

(PUBLISH_REGISTER_ERROR Message Header, Error Number, Extra Information)

3.6.6.11.6.3 When Generated

3.6.6.11.6.3.1 A PUBLISH_REGISTER_ERROR Request shall be used by the broker with the broker MAL in either the Publish Register Initiated State or Publish ReRegister Initiated State to reject a REGISTER Request.

3.6.6.11.6.3.2 The broker may reject the register attempt for one of the following reasons:

a) More providers than the broker can support shall cause a TOO_MANY error to be sent. The Extra Information field of the error contains an integer which provides the maximum number of providers supported.

b) Shutdown: the broker is shutting down. A SHUTDOWN error message shall be returned.

c) Internal error: the broker has experienced an internal error. An INTERNAL error message shall be returned.

3.6.6.11.6.3.3 A PUBLISH_REGISTER_ERROR Indication shall be generated by the provider MAL in either the Register Initiated State or ReRegister Initiated State upon one of two events:

a) the reception of a PUBLISH_REGISTER_ERROR Message, once checked via the Access Control interface, from a broker;

b) an error raised by the local communication layer.

3.6.6.11.6.4 Effect on Reception

3.6.6.11.6.4.1 Reception of a PUBLISH_REGISTER_ERROR Request shall, once checked via the Access Control interface, cause the broker MAL to transmit a PUBLISH_REGISTER_ERROR Message to the provider.

3.6.6.11.6.4.2 The pattern shall end at this point for the broker.

3.6.6.11.6.4.3 On reception of a PUBLISH_REGISTER_ERROR Message by the provider MAL, once the message has been checked via the Access Control interface, the provider MAL shall end the interaction by passing the error indication to the provider application.

3.6.6.11.6.5 Message Header

3.6.6.11.6.5.1 For the PUBLISH_REGISTER_ERROR Message, the Message Header fields shall be the same as for the PUBLISH_REGISTER_ACK Message except for the Is Error Message field, which is set to TRUE.

3.6.6.11.6.5.2 The Error Information structure shall immediately follow the Message Header.

3.6.6.11.7 PUBLISH

3.6.6.11.7.1 Function

3.6.6.11.7.1.1 The PUBLISH Request primitive shall be used by the provider to publish an entity update.

3.6.6.11.7.1.2 The PUBLISH Indication primitive shall be used by the broker MAL to deliver a PUBLISH Message to a broker.

3.6.6.11.7.2 Semantics

PUBLISH Request and PUBLISH Indication shall provide parameters as follows:

(PUBLISH Message Header, UpdateHeader, <<Update Signature>>)

3.6.6.11.7.3 When Generated

3.6.6.11.7.3.1 A PUBLISH Request shall be used by the provider with the provider MAL in the Registered State to publish an update.

3.6.6.11.7.3.2 A PUBLISH Indication shall be generated by the broker MAL in the Publish Initiated State upon reception of a PUBLISH Message, once checked via the Access Control interface, from a provider.

3.6.6.11.7.4 Effect on Reception

3.6.6.11.7.4.1 Reception of a PUBLISH Request shall, once checked via the Access Control interface, cause the provider MAL to transmit the supplied update as a PUBLISH Message to the broker.

3.6.6.11.7.4.2 On reception of a PUBLISH Message by the broker MAL, once the message has been checked via the Access Control interface, the broker MAL shall pass a PUBLISH Indication to the broker application.

3.6.6.11.7.4.3 The broker application shall match the update to the subscriptions of registered consumers that are in the Registered State using the match rules in 3.6.6.4.

3.6.6.11.7.4.4 The matched update shall be transmitted to the relevant consumers by the broker using a NOTIFY Request.

3.6.6.11.7.5 Message Header

3.6.6.11.7.5.1 For the PUBLISH Message, the Message Header fields shall be as defined in 3.5 except for the fields in table 3-38.

3.6.6.11.7.5.2 The contents of the Message Body, as specified in the operation template, shall immediately follow the Message Header.

Table 3-38: PUBLISH Message Header Fields

Field	Value
From	Provider identifier
Authentication Id	Provider Authentication Identifier
To	Broker identifier
Interaction Type	PUBSUB
Interaction Stage	5
Transaction Id	Transaction Id from PUBLISH_REGISTER Message

3.6.6.11.8 PUBLISH_ERROR

3.6.6.11.8.1 Function

3.6.6.11.8.1.1 The PUBLISH_ERROR Request primitive shall be used by the broker to reject a PUBLISH Request from a provider.

3.6.6.11.8.1.2 The PUBLISH_ERROR Indication primitive shall be used by the provider MAL to deliver a PUBLISH_ERROR Message to a provider.

3.6.6.11.8.2 Semantics

PUBLISH_ERROR Request and PUBLISH_ERROR Indication shall provide parameters as follows:

(PUBLISH_ERROR Message Header, Error Number)

3.6.6.11.8.3 When Generated

3.6.6.11.8.3.1 A PUBLISH_ERROR Request shall be used by the broker with the broker MAL in the Publish Initiated State to reject a PUBLISH Request.

3.6.6.11.8.3.2 The broker may reject the publish attempt for one of the following reasons:

a) Unknown provider: a provider has not previously registered for publishing. An INCORRECT_STATE error message shall be returned.

b) Unknown Subscription Key: a provider has attempted to publish an incorrect number of Subscription Keys. An UNKNOWN error shall be returned in this case.

c) Shutdown: the broker is shutting down. A SHUTDOWN error message shall be returned.

d) Internal error: the broker has experienced an internal error. An INTERNAL error message shall be returned.

3.6.6.11.8.3.3 Communications/Encoding/Access Control errors shall be returned to a provider using this error message. This allows a provider to receive notification of underlying errors without the publishing overhead of acknowledging each PUBLISH Message.

3.6.6.11.8.3.4 A PUBLISH_ERROR Indication shall be generated by the provider MAL in the ReRegister Initiated State, the Registered State, or the Deregister Initiated State upon one of two events:

a) the reception of a PUBLISH_ERROR Message, once checked via the Access Control interface, from a broker;

b) an error raised by the local communication layer.

3.6.6.11.8.4 Effect on Reception

3.6.6.11.8.4.1 Reception of a PUBLISH_ERROR Request shall, once checked via the Access Control interface, cause the broker MAL to transmit a PUBLISH_ERROR Message to the provider.

3.6.6.11.8.4.2 On reception of a PUBLISH_ERROR Message by the provider MAL, once the message has been checked via the Access Control interface, the provider MAL shall pass the error indication to the provider application.

3.6.6.11.8.5 Message Header

3.6.6.11.8.5.1 For the PUBLISH_ERROR Message, the Message Header fields shall be the same as for the PUBLISH Message except for the fields in table 3-39.

3.6.6.11.8.5.2 The Transaction Id shall be taken from the PUBLISH Message.

3.6.6.11.8.5.3 The Error Information shall immediately follow the Message Header.

Table 3-39: PUBLISH_ERROR Message Header Fields

Field	Value
From	Broker identifier
Authentication Id	Broker Authentication Identifier
To	Provider identifier
Is Error Message	TRUE

3.6.6.11.9 NOTIFY

3.6.6.11.9.1 Function

3.6.6.11.9.1.1 The NOTIFY Request primitive shall be used by the broker to transmit an entity update to a consumer.

3.6.6.11.9.1.2 The NOTIFY Indication primitive shall be used by the consumer MAL to deliver a NOTIFY Message to a consumer.

3.6.6.11.9.2 Semantics

NOTIFY Request and NOTIFY Indication shall provide parameters as follows:

(NOTIFY Message Header, Identifier, UpdateHeader, <<Update Signature>>)

3.6.6.11.9.3 When Generated

3.6.6.11.9.3.1 A NOTIFY Request shall be used by the broker with the broker MAL in the Registered State to transmit an entity update from a provider PUBLISH Message to a consumer.

3.6.6.11.9.3.2 A NOTIFY Indication shall be generated by the consumer MAL in the ReRegister Initiated State, the Registered State**,** or the Deregister Initiated State upon reception of a NOTIFY Message, once checked via the Access Control interface, from a broker.

3.6.6.11.9.4 Effect on Reception

3.6.6.11.9.4.1 Reception of a NOTIFY Request shall, once checked via the Access Control interface, cause the broker MAL to transmit the supplied subscription update as a NOTIFY Message to the consumer.

3.6.6.11.9.4.2 On reception of a NOTIFY Message by the consumer MAL, once the message has been checked via the Access Control interface, the consumer MAL shall pass a NOTIFY Indication to the consumer application.

3.6.6.11.9.5 Message Header

3.6.6.11.9.5.1 For the NOTIFY Message, the Message Header fields shall be as defined in 3.5 except for the fields in table 3-40.

3.6.6.11.9.5.2 The contents of the subscription update, as specified in the operation template, shall immediately follow the Message Header.

3.6.6.11.9.5.3 The Transaction Id shall be the same as in the initial REGISTER Message that created the first subscription. If several REGISTER Requests have been sent for the same subscription (without any DEREGISTER Requests in between), the Transaction Id of the first Request shall be used.

Table 3-40: NOTIFY Message Header Fields

Field	Value
From	Broker identifier
Authentication Id	Broker Authentication Identifier
To	Consumer identifier
Interaction Type	PUBSUB
Interaction Stage	6
Transaction Id	Transaction Id from REGISTER Message

3.6.6.11.10 NOTIFY_ERROR

3.6.6.11.10.1 Function

3.6.6.11.10.1.1 The NOTIFY_ERROR Request primitive shall be used by the broker to inform a consumer of an error and end a PUBSUB interaction.

3.6.6.11.10.1.2 The NOTIFY_ERROR Indication primitive shall be used by the consumer MAL to deliver a NOTIFY_ERROR Message to a consumer.

3.6.6.11.10.2 Semantics

NOTIFY_ERROR Request and NOTIFY_ERROR Indication shall provide parameters as follows:

(NOTIFY_ERROR Message Header, Error Number)

3.6.6.11.10.3 When Generated

3.6.6.11.10.3.1 A NOTIFY_ERROR Request shall be used by the broker with the broker MAL in the Registered State to inform a consumer of an error.

3.6.6.11.10.3.2 The broker may send an error for one of the following reasons:

a) Shutdown: the broker is shutting down. A SHUTDOWN error message shall be returned.

b) Internal error: the broker has experienced an internal error. An INTERNAL error message shall be returned.

3.6.6.11.10.3.3 A NOTIFY_ERROR Indication shall be generated by the consumer MAL in the ReRegister Initiated State, the Registered State, or the Deregister Initiated State upon one of two events:

a) the reception of a NOTIFY_ERROR Message, once checked via the Access Control interface, from a broker;

b) an error raised by the local communication layer.

3.6.6.11.10.4 Effect on Reception

3.6.6.11.10.4.1 Reception of a NOTIFY_ERROR Request shall, once checked via the Access Control interface, cause the broker MAL to transmit a NOTIFY_ERROR Message to the consumer.

3.6.6.11.10.4.2 The pattern shall end at this point for the broker.

3.6.6.11.10.4.3 On reception of a NOTIFY_ERROR Message by the consumer MAL, once the message has been checked via the Access Control interface, the consumer MAL shall end the interaction by passing the error to the consumer application.

3.6.6.11.10.5 Message Header

3.6.6.11.10.5.1 For the NOTIFY_ERROR Message, the Message Header fields shall be the same as for the NOTIFY Message except for the Is Error Message field, which is set to TRUE.

3.6.6.11.10.5.2 The Error Information shall immediately follow the Message Header.

3.6.6.11.11 DEREGISTER

3.6.6.11.11.1 Function

3.6.6.11.11.1.1 The DEREGISTER Request primitive shall be used by the consumer to end subscriptions for the specified subscription identifiers.

3.6.6.11.11.1.2 The DEREGISTER Indication primitive shall be used by the broker MAL to deliver a DEREGISTER Message to a broker and end existing subscriptions for the specified subscription identifiers.

3.6.6.11.11.2 Semantics

DEREGISTER Request and DEREGISTER Indication shall provide parameters as follows:

(DEREGISTER Message Header, List<Identifier>)

3.6.6.11.11.3 When Generated

3.6.6.11.11.3.1 A DEREGISTER Request shall be used by the consumer application with the consumer MAL in the Registered State at any time to end a subscription.

3.6.6.11.11.3.2 A DEREGISTER Indication shall be generated by the broker MAL in the Registered State upon reception of a DEREGISTER Message, once checked via the Access Control interface, from a consumer.

3.6.6.11.11.4 Effect on Reception

3.6.6.11.11.4.1 Reception of a DEREGISTER Request shall, once checked via the Access Control interface, cause the consumer MAL in the Registered State to end the set of subscriptions by transmitting a DEREGISTER Message to the broker.

3.6.6.11.11.4.2 The consumer MAL shall enter the Deregister Initiated State for each identified subscription.

3.6.6.11.11.4.3 On reception of a DEREGISTER Message by the broker MAL, once the message has been checked via the Access Control interface, the broker MAL shall enter the Deregister Initiated State for each of the identified subscriptions and pass a DEREGISTER Indication to the broker.

3.6.6.11.11.4.4 The broker shall then remove the specified subscriptions of the consumer.

3.6.6.11.11.5 Message Header

3.6.6.11.11.5.1 For the DEREGISTER Message, the Message Header fields shall be as defined in 3.5 except for the fields in table 3-41.

3.6.6.11.11.5.2 The contents of the subscription identifier list, as specified in the operation template, shall immediately follow the Message Header.

Table 3-41: DEREGISTER Message Header Fields

Field	Value
From	Consumer identifier
Authentication Id	Consumer Authentication Identifier
To	Broker identifier
Interaction Type	PUBSUB
Interaction Stage	7
Transaction Id	Transaction Id from consumer MAL

3.6.6.11.12 DEREGISTER_ACK

3.6.6.11.12.1 Function

3.6.6.11.12.1.1 The DEREGISTER_ACK Request primitive shall be used by the broker to acknowledge the termination of a PUBSUB interaction subscription by the consumer.

3.6.6.11.12.1.2 The DEREGISTER_ACK Indication primitive shall be used by the consumer MAL to deliver a DEREGISTER_ACK Message to a consumer.

3.6.6.11.12.2 Semantics

DEREGISTER_ACK Request and DEREGISTER_ACK Indication shall provide parameters as follows:

(DEREGISTER_ACK Message Header)

3.6.6.11.12.3 When Generated

3.6.6.11.12.3.1 A DEREGISTER_ACK Request shall be used by the broker with a broker MAL in the Deregister Initiated State to acknowledge the successful deregistration by a consumer of a set of subscriptions in a PUBSUB interaction.

3.6.6.11.12.3.2 A DEREGISTER_ACK Indication shall be generated by the consumer MAL in the Deregister Initiated State upon reception of a DEREGISTER_ACK Message, once checked via the Access Control interface, from a broker.

3.6.6.11.12.4 Effect on Reception

3.6.6.11.12.4.1 Reception of a DEREGISTER_ACK Request shall, once checked via the Access Control interface, cause the broker MAL to transmit a DEREGISTER_ACK Message to the consumer.

3.6.6.11.12.4.2 The interaction shall end at this point for the broker for each of the identified subscriptions.

3.6.6.11.12.4.3 On reception of a DEREGISTER_ACK Message by the consumer MAL in the Deregister Initiated State, once the message has been checked via the Access Control interface, the consumer MAL shall pass the DEREGISTER_ACK Indication to the consumer application.

3.6.6.11.12.4.4 The interaction shall end here for the consumer for the set of subscriptions in the initiating DEREGISTER Message.

3.6.6.11.12.5 Message Header

For the DEREGISTER_ACK Message, the Message Header fields shall be as defined in 3.5 except for the fields in table 3-42.

Table 3-42: DEREGISTER_ACK Message Header Fields

Field	Value
From	Broker identifier
Authentication Id	Broker Authentication Identifier
To	Consumer identifier
Interaction Type	PUBSUB
Interaction Stage	8
Transaction Id	Transaction Id from DEREGISTER Message

3.6.6.11.13 PUBLISH_DEREGISTER

3.6.6.11.13.1 Function

3.6.6.11.13.1.1 The PUBLISH_DEREGISTER Request primitive shall be used by the provider to end a PUBSUB interaction.

3.6.6.11.13.1.2 The PUBLISH_DEREGISTER Indication primitive shall be used by the broker MAL to deliver a PUBLISH_DEREGISTER Message to a broker and end a PUBSUB interaction for the specified provider.

3.6.6.11.13.2 Semantics

PUBLISH_DEREGISTER Request and PUBLISH_DEREGISTER Indication shall provide parameters as follows:

(PUBLISH_DEREGISTER Message Header)

3.6.6.11.13.3 When Generated

3.6.6.11.13.3.1 A PUBLISH_DEREGISTER Request shall be used by the provider application with the provider MAL in the Registered State to end the interaction.

3.6.6.11.13.3.2 A PUBLISH_DEREGISTER Indication shall be generated by the broker MAL in the Publish Initiated State upon reception of a PUBLISH_DEREGISTER Message, once checked via the Access Control interface, from a provider.

3.6.6.11.13.4 Effect on Reception

3.6.6.11.13.4.1 Reception of a PUBLISH_DEREGISTER Request shall, once checked via the Access Control interface, cause the provider MAL in the Registered State to request the end of the interaction by transmitting a PUBLISH_DEREGISTER Message to the broker.

3.6.6.11.13.4.2 The provider MAL shall enter the Deregister Initiated State.

3.6.6.11.13.4.3 On reception of a PUBLISH_DEREGISTER Message by the broker MAL, once the message has been checked via the Access Control interface, the broker MAL shall enter the Publish Deregister Initiated State.

3.6.6.11.13.4.4 The broker shall then remove the provider from the list of allowed publishers.

3.6.6.11.13.5 Message Header

For the PUBLISH_DEREGISTER Message, the Message Header fields shall be as defined in 3.5 except for the fields in table 3-43.

Table 3-43: PUBLISH_DEREGISTER Message Header Fields

Field	Value
From	Provider identifier
Authentication Id	Provider Authentication Identifier
To	Broker identifier
Interaction Type	PUBSUB
Interaction Stage	9
Transaction Id	Transaction Id from provider MAL

3.6.6.11.14 PUBLISH_DEREGISTER_ACK

3.6.6.11.14.1 Function

3.6.6.11.14.1.1 The PUBLISH_DEREGISTER_ACK Request primitive shall be used by the broker to acknowledge the termination of a PUBSUB interaction by the provider.

3.6.6.11.14.1.2 The PUBLISH_DEREGISTER_ACK Indication primitive shall be used by the provider MAL to deliver a PUBLISH_DEREGISTER_ACK Message to a provider.

3.6.6.11.14.2 Semantics

PUBLISH_DEREGISTER_ACK Request and PUBLISH_DEREGISTER_ACK Indication shall provide parameters as follows:

(PUBLISH_DEREGISTER_ACK Message Header)

3.6.6.11.14.3 When Generated

3.6.6.11.14.3.1 A PUBLISH_DEREGISTER_ACK Request shall be used by the broker with the broker MAL in the Publish Deregister Initiated State to acknowledge the successful deregistration by a provider in a PUBSUB interaction.

3.6.6.11.14.3.2 A PUBLISH_DEREGISTER_ACK Indication shall be generated by the provider MAL in the Deregister Initiated State upon reception of a PUBLISH_DEREGISTER_ACK Message, once checked via the Access Control interface, from a broker.

3.6.6.11.14.4 Effect on Reception

3.6.6.11.14.4.1 Reception of a PUBLISH_DEREGISTER_ACK Request shall, once checked via the Access Control interface, cause the broker MAL to transmit the supplied acknowledgement as a PUBLISH_DEREGISTER_ACK Message to the provider.

3.6.6.11.14.4.2 The interaction shall end at this point for the broker.

3.6.6.11.14.4.3 On reception of a PUBLISH_DEREGISTER_ACK Message by the provider MAL in the Deregister Initiated State, once the message has been checked via the Access Control interface, the provider MAL shall pass the PUBLISH_DEREGISTER_ACK Indication to the provider application.

3.6.6.11.14.4.4 The interaction shall end here for the provider.

3.6.6.11.14.5 Message Header

For the PUBLISH_DEREGISTER_ACK Message, the Message Header fields shall be as defined in 3.5 except for the fields in table 3-44.

Table 3-44: PUBLISH_DEREGISTER_ACK Message Header Fields

Field	Value
From	Broker identifier
Authentication Id	Broker Authentication Identifier
To	Provider identifier
Interaction Type	PUBSUB
Interaction Stage	10
Transaction Id	Transaction Id from PUBLISH_DEREGISTER Message

3.6.6.12 Discussion

The following example shows a simple example service that contains the following PUBLISH-SUBSCRIBE operation:

Operation Identifier	testPubSub		
Interaction Pattern	PUBLISH-SUBSCRIBE		
Subscription Keys	Identifier key1 Long key2		
Pattern Sequence	Message	Nullable	Type Signature
OUT	PUBLISH/NOTIFY	Yes No Yes	Boolean field1 Integer field2 TestNotify field3

The TestNotify structure is defined below:

Name	TestNotify		
Extends	Composite		
Short Form Part	Example Only		
Field	Type	Nullable	Comment
name	Identifier	Yes	Example Identifier item
value	Integer	Yes	Example Integer item

To register for this PUBLISH-SUBSCRIBE operation, a consumer could send the following message to the broker:

Message Header												
From	Authentication Id	To	Timestamp	Interaction Type	Interaction Stage	Transaction Id	Service Area	Service	Operation	Area Version	Is Error Message	Supplements
Consumer	SC X	Broker		PUBSUB	1	123	Example	Example	testPubSub	1	FALSE	

Subscription							
			filters				
			List<SubscriptionFilter>				
subscriptionId	domain	selectedKeys	List count	SubscriptionFilter			
				name	values		
					List count	value	value
Sub123	AA.BB	NULL	1	key2	2	45	100

This would result in the following acknowledgement message being sent back:

Message Header												
From	Authentication Id	To	Timestamp	Interaction Type	Interaction Stage	Transaction Id	Service Area	Service	Operation	Area Version	Is Error Message	Supplements
Broker	BRQ	Consumer		PUBSUB	2	123	Example	Example	testPubSub	1	FALSE	

To publish an update, a previously registered provider sends the following message to the broker:

Message Header												
From	Authentication Id	To	Timestamp	Interaction Type	Interaction Stage	Transaction Id	Service Area	Service	Operation	Area Version	Is Error Message	Supplements
Consumer	SC X	Broker		PUBSUB	5	124	Example	Example	testPubSub	1	FALSE	

UpdateHeader					field1	field2	field3	
		keyValues					TestNotify	
source	domain	List<Attribute>			Boolean	Integer	name	value
		List count	value	value			Identifier	Integer
SC X	A.B	2	param1	6	FALSE	33	text	1234

When an update is required, the following NOTIFY Message is sent from the message broker to the consumer:

Message Header												
From	Authentication Id	To	Timestamp	Interaction Type	Interaction Stage	Transaction Id	Service Area	Service	Operation	Area Version	Is Error Message	Supplements
Broker	BR Q	Consumer		PUBSUB	6	123	Example	Example	testPubSub	1	FALSE	

subscriptionId	UpdateHeader					field1	field2	field3	
			keyValues					TestNotify	
Identifier	source	domain	List<Attribute>			Boolean	Integer	name	value
			List count	value	value			Identifier	Integer
Sub 123	SC X	A.B	2	param1	6	FALSE	33	text	1234

No acknowledgment of the NOTIFY Message is sent by the consumer.

When the consumer wants to deregister, the following message is sent:

Message Header												
From	Authentication Id	To	Timestamp	Interaction Type	Interaction Stage	Transaction Id	Service Area	Service	Operation	Area Version	Is Error Message	Supplements
Consumer	Op A	Broker		PUBSUB	7	125	Example	Example	testPubSub	1	FALSE	

List<Identifier>	
List count	Id
1	Sub 123

This results in the following acknowledgment being received:

Message Header												
From	Authentication Id	To	Timestamp	Interaction Type	Interaction Stage	Transaction Id	Service Area	Service	Operation	Area Version	Is Error Message	Supplements
Broker	BR Q	Consumer		PUBSUB	8	125	Example	Example	testPubSub	1	FALSE	

NOTE – An actual service specification, rather than the example given here, would fully specify the meaning of, and required values of, all structures used by the service.

3.7 ACCESS CONTROL INTERFACE

3.7.1 OVERVIEW

The MAL specification does not make any assumption on the applicable security requirements for a concrete deployment. To this end, the specification provides a generic Access Control Interface to formalize the interactions with any deployment specific Access Control component.

The security requirements of concrete deployments may vary from none to extremely stringent confidentiality, integrity, and availability requirements. For many (legacy and still-operational) space missions, no authentication, authorization, or assurances for confidentiality of the communication between the space and ground have been implemented. For many other space missions, various levels of security requirements have been made applicable to the space-to-ground interface as well as to ground-to-ground interactions (e.g., between control centres).

Hence the MAL specification needs to support the wide spectrum of scenarios in which a simple service may broadcast non-sensitive information to the general public, as well as scenarios with elaborate certificate-based authentications, rule- and content-based authorizations, and end-to-end encryption of communication with sophisticated key management systems.

The MO Reference Model (reference [1]) and the Mission Operations Concept Green Book (reference [D1]) provide a more detailed consideration of the security aspects and recommended security profiles of typical deployment scenarios for space missions.

The Authentication Id in the MAL header supports accordingly the full ranges of authentication means from none, to a simple text-based user name and password, up to token or certificate-based authentications. The MAL is positioned between the Transport Layer and the Application Layer. Upon reception of any message from the Transport Layer and before reaching up to the Application Layer, MAL passes every MAL message through this specified Access Control interface to the Access Control component. It is the delegated responsibility of the deployment-specific implementation of an Access Control component to ensure the enforcement of the necessary security assurances for a particular deployment using the information provided in the Authentication Id field of every MAL message and the message itself. Only upon a positive check response from the Access Control component through the standardized Access Control Interface, MAL will pass the message to the Application Layer.

3.7.2 CHECK MESSAGE INTERACTION

3.7.2.1 Overview

3.7.2.1.1 General

The CHECK pattern is similar to the REQUEST Interaction Pattern: a MAL message is passed in, and the Access Control component responds with a return MAL message. No acknowledgement other than the response is sent:

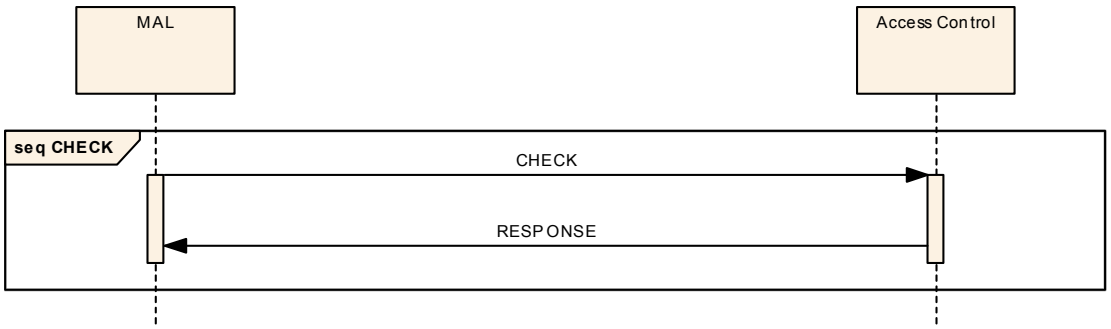

Figure 3-28: CHECK Access Control Pattern Message Sequence

3.7.2.1.2 Description

The CHECK pattern provides a simple request/response message exchange that is used by a MAL implementation to perform Access Control.

NOTE – It is not expected that this pattern will be implemented via a message transport, as it is expected to be implemented as a local API used by the MAL. It is shown here as a pattern for consistency.

3.7.2.2 Usage

3.7.2.2.1 The CHECK pattern shall be used only by the MAL for the checking of incoming and outgoing messages. It is not used by any other component.

3.7.2.2.2 A compliant MAL implementation shall follow the sequences defined in sections 4 and 5 of reference [1] for interacting with an Access Control component.

3.7.2.2.3 A broker in a Publish Subscribe pattern shall also submit any NOTIFY and PUBLISH Messages to its Access Control component.

3.7.2.2.4 Access to sensitive data distributed via the PUBSUB pattern shall be filtered at this point if required.

NOTE – For the restriction of sensitive data, it is legitimate for a broker to integrate the CHECK logic into the subscription matching logic.

3.7.2.3 Error Handling

3.7.2.3.1 The response shall be replaced with an error message (see section 4) if an error occurs during the processing of the operation (figure 3-29).

3.7.2.3.2 Either the response or the error message shall be returned, but never both.

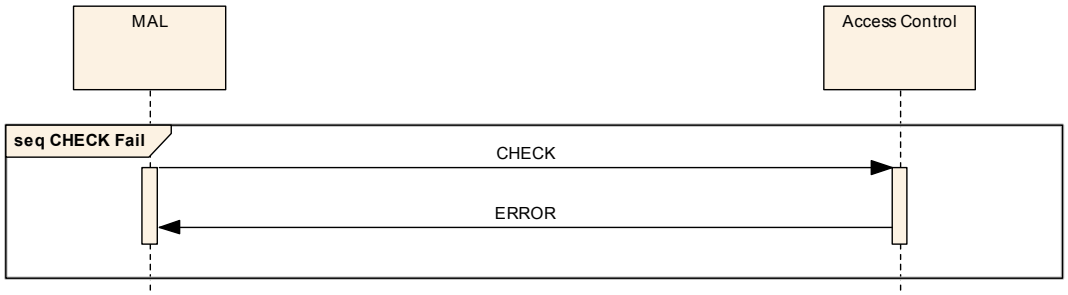

Figure 3-29: CHECK Access Control Pattern Error Sequence

3.7.2.4 Operation Template

The CHECK pattern shall conform to the CHECK operation template defined in table 3-45.

Table 3-45: CHECK Operation Template

Interaction Pattern	CHECK		
Pattern Sequence	Message	Nullability	Type Signature
IN	CHECK	N/A	MAL message
OUT	RESPONSE	N/A	MAL message

3.7.2.5 Primitives

The CHECK pattern shall use the primitives defined in table 3-46.

Table 3-46: CHECK Primitive List

Primitive
CHECK
RESPONSE
ERROR

NOTE – The CHECK pattern is not an abstract interface that provides interoperability. However, it is a function that is required by the MAL.

3.7.2.6 State Charts

No state charts are provided for this pattern. The MAL shall initiate the pattern using the CHECK primitive and shall block until either the Response or Error Indication is received.

3.7.2.7 Requests and Indications

3.7.2.7.1 CHECK

3.7.2.7.1.1 Function

The CHECK Request primitive shall be used by the MAL to initiate a CHECK interaction.

3.7.2.7.1.2 Semantics

CHECK Request shall provide parameters as follows:

(MAL Message)

3.7.2.7.1.3 When Generated

3.7.2.7.1.3.1 A CHECK Request shall be generated by the MAL after reception of a message from either the application or from a transport.

3.7.2.7.1.3.2 The checks an implementation of the Access Control component shall perform upon reception of a CHECK Request are implementation and deployment specific.

3.7.2.7.2 RESPONSE

3.7.2.7.2.1 Function

The RESPONSE Indication primitive shall be used by the Access Control component to deliver a RESPONSE Message to the MAL.

3.7.2.7.2.2 Semantics

RESPONSE Indication shall provide parameters as follows:

(MAL Message)

3.7.2.7.2.3 When Generated

A RESPONSE Indication shall be generated upon reception of a RESPONSE Message from the Access Control component.

3.7.2.7.2.4 Effect on Reception

3.7.2.7.2.4.1 On reception of a RESPONSE Indication, the MAL shall end the Interaction Pattern with success.

3.7.2.7.2.4.2 The returned message shall then be used by the MAL from that point onwards.

3.7.2.7.3 ERROR

3.7.2.7.3.1 Function

The ERROR Indication primitive shall be used by the Access Control component to end a CHECK interaction with an error.

3.7.2.7.3.2 Semantics

ERROR Indication shall provide parameters as follows:

(Error Number, Extra Information)

3.7.2.7.3.3 When Generated

3.7.2.7.3.3.1 An ERROR Indication shall be generated upon reception of an ERROR Message from the Access Control component.

3.7.2.7.3.3.2 The Access Control component shall return AUTHENTICATION_FAIL if the supplied message fails authentication checks. These checks are implementation and deployment specific.

3.7.2.7.3.3.3 The Access Control component shall return AUTHORISATION_FAIL if the authenticated supplied message fails authorization checks. These checks are implementation and deployment specific.

3.7.2.7.3.4 Effect on Reception

3.7.2.7.3.4.1 On reception of an ERROR Indication, the MAL shall end the interaction with an error.

3.7.2.7.3.4.2 If the Access Control error is to be transmitted to another MAL, then the resultant ERROR Message shall not be passed to the Access Control component, as it originated from that.

3.7.2.7.3.4.3 The behaviour of the MAL from this point forward is defined in reference [1].

4 MAL DATA TYPE SPECIFICATION

4.1 OVERVIEW

The specification of the abstract interfaces and services details the structures passed as the message bodies and message returns of the Interaction Patterns and operations. This section details the MAL data type specification, the rules for its use with the MAL, the types and structures it contains, and the rules allowed for the combination of these.

Figure 4-1 shows the MAL data model in UML class diagram notation. MAL-defined types are shown with grey background, examples of service-defined data types are shown with white background. Generalization relationship arrows point from the specialized type to the general type. Abstract types are denoted by the '{abstract}' property. Lists have a dashed border because they are not defined explicitly. The ObjectRef attribute refers to a concrete MO Object of type 'SomeObj' just as an example. From all Composites defined by the MAL, only the ObjectIdentity composite is shown explicitly because of its special role for MO Objects as their identity type. Enumerations, Lists, and MO Objects are denoted using the appropriate keyword in guillemets («, »). Ellipses (…) are used to indicate that the types or fields depicted here are not exhaustive.

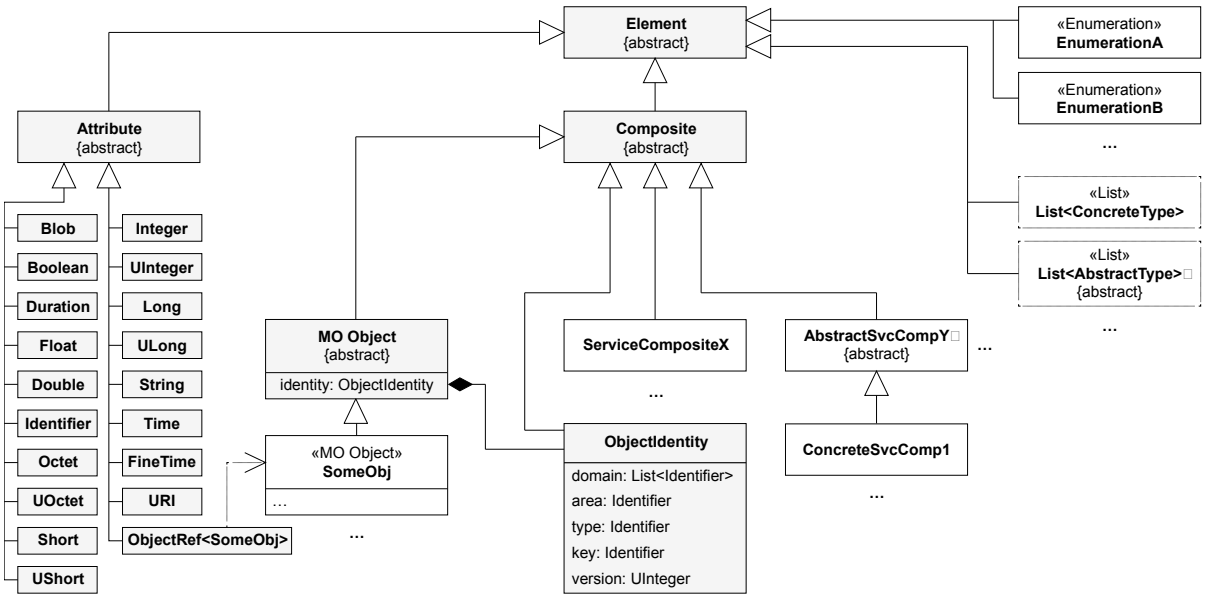

Figure 4-1: MAL Data Types Example

Other data type specification languages (such as XML schema; see reference [6]) may be used to specify the message bodies in service specifications; however, support for these other data type specification languages in a particular technology mapping may not be universal and may limit use of a specification to specific deployments.

Each Area can define its own data types, derived from the MAL types and composition rules defined in this document. Types shall be defined at the Area-level only.

4.2 REPRESENTATIONS

4.2.1 REPRESENTING ABSTRACT AND CONCRETE TYPES

The MAL type specification allows the definition of two basic categories of types: abstract and concrete.

Abstract types are represented in a table as illustrated below:

Name	<<Name>>
Extends	<<Extension Type Name>>
Abstract	

The name of the type is given in the Name field, and the name of the abstract type that this type extends is given in the Extends field.

Concrete types are represented in a table as illustrated below:

Name	<<Name>>
Extends	<<Extension Type Name>>
Short Form Part	<<Short Form Part>>

Each concrete type is defined with a numerical Short Form Part that is expected to be used by efficient encodings when required to hold type information. Each specification that defines types shall number each concrete type in the appropriate field starting from 1.

NOTE – Abstract types cannot be instantiated at runtime. If a service specification specifies a certain type as abstract and specifies its usage in a service operation or in a Composite structure, then, at runtime, a concrete type derived from the specified abstract shall be used. On the wire, the part where the specified abstract type was defined will have a concrete type derived from it.

To avoid number conflicts between types defined in different areas, a fully qualified Short Form Part is defined as ‘Type Id’. The ‘Type Id’ is composed of 4 parts:

a) the area number (as a UShort);

b) the area version (as a UShort);

c) the service number (as a UShort);

d) the Short Form Part from the type definition (as a Short).

All types are defined at Area-level and therefore the service number shall be set to zero. The service number may be used for backward-compatible implementations.

In the context of the MAL specification, the area number of ‘1’ is used.

4.2.2 REPRESENTING FUNDAMENTALS

The base type for all types and structures is Element. Three other Fundamental types exist: Composite, Attribute, and Object. Only the MAL specification (this document) is allowed to define Fundamental types.

Fundamental types are represented in a table as illustrated below:

Name	Element
Extends	
Abstract	

4.2.3 REPRESENTING ATTRIBUTES

Attributes are the simplest MAL type; they cannot be decomposed into any smaller Elements and are used to build more complex structures.

Only the MAL specification (this document) is allowed to define Attribute types.

Attribute types are represented in a table as illustrated below:

Name	<<Attribute Name>>
Extends	Attribute
Short Form Part	<<Short Form Part>>

All Attribute types extend the Fundamental type ‘Attribute’. They are also defined in 4.3.4, but the actual representation, or encoding, of them is completely dependent on the language and transport/encoding mapping used. However, because the limits and behaviour of the types are constant (defined here), the informational content is preserved when moving between mappings.

For example, a Boolean value may be represented by a single bit in some encodings or by the text strings ‘True/False’ in others; however, the meaning of the value is identical regardless of the encoding used.

4.2.4 REPRESENTING COMPOSITES

4.2.4.1 General

Composites are represented in a table as illustrated below:

Name	TestBody		
Extends	Composite		
Short Form Part	123		
Field	Type	Nullable	Comment
FirstItem	String	Yes	Example String item.
SecondItem	Integer	Yes	Example Integer item.

The Nullable column indicates whether the field is permitted to contain a NULL value. If the field is set to Yes, then it is allowed to contain a NULL value; if it is set to No, then it is not.

4.2.4.2 Polymorphism—Composite Extension

Composite structures can be extended, with the following conditions:

- an abstract Composite can be extended by another abstract Composite;
- an abstract Composite can be extended by a concrete Composite;
- a Composite, either concrete or abstract, can only extend one unique abstract Composite; therefore a Composite cannot extend multiple Composites at the same time;
- a Composite extending an abstract Composite cannot specify fields with the same name that are already specified in the extended abstract Composite;
- a concrete Composite may extend an abstract Composite without specifying new fields;
- a concrete Composite cannot be extended further.

NOTE – It is not permitted to extend a concrete Composite further, as an issue arises when the extended Composite is used in place of the base Composite (which is not marked as abstract). Because the receiving application is given no indication that the message contains extra information, it is not possible to decode the message correctly. Some transport encodings may be able to support this, but many will not be able to. Therefore only abstract Composites shall be extended.

A Composite extending a base abstract Composite shall be represented as any other Composite with a row for all of its fields, including the fields inherited from the base abstract Composites. The rows representing the inherited fields are filled with a grey background.

The following structures present one example of an abstract Composite being extended by a concrete Composite.

Name	AbstractComposite		
Extends	Composite		
Abstract			
Field	Type	Nullable	Comment
FirstItem	String	Yes	Example String item.
SecondItem	Integer	Yes	Example Integer item.

Name	ComplexStructure		
Extends	**AbstractComposite**		
Short Form Part	345		
Field	Type	Nullable	Comment
FirstItem	String	Yes	Example String item.
SecondItem	Integer	Yes	Example Integer item.
third_item	Boolean	Yes	Extra Boolean item.
fourth_item	Integer	Yes	Example Integer item.
last_item	TestBody	Yes	Contained structure.

This shows that ComplexStructure can be considered an extension of AbstractComposite and contains all the contents of AbstractComposite plus its own extra items.

Good practices for service design:

A sound architectural design favours the 'composition over inheritance' principle (or Composite reuse principle). This principle states that polymorphic behaviour and code reuse should be achieved by their composition (by containing other Composite structures) rather than inheritance from a base or parent structure. This is an often-stated principle in object-oriented programming. MO Architects are advised to take this principle into consideration when defining new services.

4.2.4.3 Polymorphism—Abstract Elements in Composites

The type of the fields within a Composite can be specified as an abstract or as a concrete type. This includes not only the abstract Composites but also the abstract Attribute type.

NOTE – By defining an abstract field within a Composite, the Composite itself does not become abstract.

For example, the following is permitted:

Name	ConcreteCompositeA		
Extends	Composite		
Short Form Part	654321		
Field	Type	Nullable	Comment
SomeItem	Attribute	Yes	Example Attribute item.

And the following is also permitted:

Name	ConcreteCompositeB		
Extends	Composite		
Short Form Part	123456		
Field	Type	Nullable	Comment
AnotherItem	ComplexStructure	Yes	Example of extended abstract type.
FurtherItem	AbstractComposite	Yes	Example abstract type.

4.2.5 REPRESENTING ENUMERATIONS

4.2.5.1.1 Enumerations shall define sets of possible values.

4.2.5.1.2 All Enumerations shall be extensions of the Fundamental Element type and therefore may be used to replace an abstract Element type in message bodies.

4.2.5.1.3 Enumerations shall be represented as follows:

Name	ExampleEnum	
Short Form Part	789	
Enumeration Value	Numeric Value	Comment
FIRST	1	First enumeration possible value.
SECOND	2	Second enumeration value.
OTHER	3	Etc.
LAST	4	Last enumeration value.

4.2.5.1.4 Each Enumeration Value shall have an associated Numeric Value with a value in the UShort range. It is expected to be used in efficient encodings and transport mappings.

NOTE – Implementations might use Enumeration Values outside the defined ones for a certain Enumeration. In this case, an out-of-band agreement would be required for the additional values.

4.2.6 REPRESENTING LISTS

4.2.6.1 General

4.2.6.1.1 A List shall be an arbitrary-length sequence of items.

4.2.6.1.2 Lists may be created of any Concrete or Abstract type (including Element, Attribute, Enumeration, Composite, or Object).

4.2.6.1.3 All Lists shall be extensions of the Fundamental Element type and therefore may be used to replace an abstract Element type in message bodies.

4.2.6.1.4 Lists shall be represented in the specifications as follows:

List<ExampleStructure>

The List above is shown as having a ‘type’ of ExampleStructure and is therefore a list of ExampleStructures.

NOTE – A List has a length part, but whether an actual length field is required depends on the selected encoding. For example, a binary encoding can include an initial length field to allow correct decommutation, whereas an XML-based encoding would not require this because XML tags denote the end of the List (reference [6]).

4.2.6.1.5 If the type of the List is Concrete, then the List shall be a Homogeneous List.

4.2.6.1.6 If the type of the List is Abstract, then the List shall be a Heterogeneous List.

4.2.6.2 Homogeneous List

4.2.6.2.1 The Short Form Part of a Homogeneous List shall be set as the negative Short Form Part of the type defined for the list.

NOTE – For example, the Short Form Part of the Integer type is ‘11’, therefore the Short Form Part of a list of Integers will be ‘−11’.

4.2.6.3 Heterogeneous List

4.2.6.3.1 The Short Form Part of a Heterogenous List shall be set as '0'.

NOTE – A Heterogenous List can hold different Concrete types that extend the selected base Abstract type. An example of a Heterogenous List is presented:

- Elements (List<AbstractBaseType>)
  - Element [0]—ConcreteTypeA
  - Element [1]—ConcreteTypeB
  - Element [2]—ConcreteTypeA
  - Element [3]—ConcreteTypeC

NOTE – If a Heterogenous List is specified, some encoders may have to include the 'Type Id' for each of the Elements in that List in order to be able to decode them correctly. This is, however, encoder-specific.

4.2.7 REPRESENTING OBJECT REFERENCES

4.2.7.1 An object reference is a reference to an instance of an MO Object (defined in 4.4). Object references use the ObjectRef Attribute type as defined in 4.5.19. When specifying an object reference, the reference shall refer to an MO Object (which is defined using the MAL Object type). Object references are represented in the service specifications as below:

ObjectRef<ExampleStructure>

NOTE – The object reference is shown as having a 'type' of ExampleStructure and is therefore a reference to an MO Object of ExampleStructure type.

4.2.7.2 If the object reference type is unknown or kept open to different types of MO Objects deliberately, then the representation shall use the MAL base type Element as follows:

ObjectRef<Element>

NOTE – Any programming language mapping is free to implement type checking rules between the declared Object type of the reference and the actual Object type of the reference value. This is, however, implementation-specific.

4.2.8 REPRESENTING NULL

In some message structures, it may be required to have optional components. To this end, it is required to be able to represent, for each type of component, the concept of NULL. This is separate and completely different from the concept of empty. For example, an empty string (‘’) is different from a NULL string which has no value. Language mappings must have the ability to represent NULL in messages, and transport mappings must have the ability to transport NULL.

Each Composite structure definition contains an entry for each field denoting whether that field shall be allowed to contain the NULL value.

4.2.9 REPRESENTING NULLABILITY

As presented in the previous section, the concept of NULL exists and can be represented. Moreover, nullability is the ability of a field to hold a NULL value. This is a field property that must also be representable.

In Composite structures, the nullability property of each field is shown in the ‘Nullable’ column of the Composite representation tables. Similarly, in the service operations, the nullability property of each field is shown in the ‘Nullable’ column of the representation table for each operation.

NOTE – The nullability property in fields is important and should not be disregarded when specifying a service. In some encodings, this will change how the data is encoded, which might directly translate into low-level performance optimizations.

A summary of the nullability property in the different types of data follows:

- Composite fields—selectable;
- Service operation fields—selectable;
- PUB-SUB PUBLISH message:
  - Update Header field—non-nullable,
  - Update Signature fields—selectable;
- PUB-SUB NOTIFY message:
  - SubscriptionId field—non-nullable,
  - Update Header field—non-nullable,
  - Update Signature fields—selectable;
- Entries in Lists—non-nullable.

4.3 FUNDAMENTALS

4.3.1 ELEMENT

Element is the base type of all data constructs. All types that make up the MAL data model are derived from it.

Name	Element
Extends	
Abstract	

4.3.2 ATTRIBUTE

Attribute is the base type of all Attributes of the MAL data model. Attributes are contained within Composites and are used to build complex structures that make the data model.

Name	Attribute
Extends	Element
Abstract	

4.3.3 COMPOSITE

Composite is the base structure for Composite structures that contain a set of Elements.

Name	Composite
Extends	Element
Abstract	

4.3.4 OBJECT

Object is the base structure for MO Objects in the MAL data model. Objects are representations of complex data types with two specific characteristics: Objects have a unique and immutable identity and can be referenced unambiguously.

Name	Object		
Extends	Composite		
Abstract			
Field	Type	Nullable	Comment
identity	ObjectIdentity	No	The MO Object Identity.

4.4 MO OBJECTS

4.4.1 OVERVIEW

MO Objects are representations of complex data types with two specific characteristics: MO Objects have a unique and immutable identity and can be referenced unambiguously, using the MAL ObjectRef type (see 4.5.19). MO Objects are used to represent and formally specify the complex data model in service specifications.

The MAL Object type is an extension of the Composite type. All MO Objects are to be modelled as specializations of the MAL Object type (see 4.3.4). The identity of an MO Object is represented in more detail in the next section.

MO Objects can be explicitly referenced by other Composite data structures with the use of the ObjectRef MAL data type. The ObjectRef data type is described in 4.5.19.

4.4.2 OBJECT IDENTITY

4.4.2.1 Each MO Object shall have a unique and immutable Object Identity (defined in 4.6.13).

4.4.2.2 The Object Identity shall be composed of a Domain, a Key, and an Object Version, as defined in table 4-1.

NOTE – This combination allows the MO Objects to be uniquely identified across the different domains and areas. The Object Version field of the Object Identity is not the same as the area version. The Object Version field enables versioning of MO Objects, while the area version is the version of the area that was assigned.

Table 4-1: Object Identity Fields

Field	Type	Value
Domain	List<Identifier>	Domain of the MO Object
Key	Identifier	Key of the MO Object
Object Version	UInteger	Version of the MO Object, starting at 1

NOTE – An ObjectIdentity MAL Composite is defined in this Recommended Standard, and it includes the exact same fields of the Object Identity as presented above. This allows its direct usage in MAL Composite data structures in order to represent the identity of a certain MO Object.

4.4.2.3 The Domain field shall be set according to the requirements in 3.2.2.

4.4.2.4 The Key field shall be unique within the scope of the Domain, for a given area and type.

4.4.2.5 If an MO Object does not have versions, then the Object Version field shall be populated with a '1'.

4.4.2.6 Value '0' for the version of the Object Identity is reserved to reference to the latest version of the MO Object and shall not be used for versioning purposes.

4.4.2.7 The Object Identity shall be used to represent the identity of an MO Object in an MAL Object type and not to make references to other MO Objects. In order to reference an MO Object, the ObjectRef type shall be used.

NOTE – A provider may decide to delete an MO Object, making its assigned Object Identity no longer useful. Afterwards, the provider might or might not reuse the same Object Identity to uniquely identify a newly created MO Object. This is an implementation-specific decision.

4.4.3 REPRESENTATION EXAMPLE

As previously defined, an MO Object can be defined using the MAL Object data type. The following table presents an example of how an MO Object is expected to be represented in the specifications.

The MO Object is called ExampleStructure and includes a first field with an ObjectIdentity MAL type, and two additional example fields.

Name	ExampleStructure		
Extends	Object		
Short Form Part	123		
Field	Type	Nullable	Comment
identity	ObjectIdentity	No	The identity of this MO Object.
myField1	Integer	Yes	An example of a field in the MO Object.
myField2	String	Yes	An example of a field in the MO Object.

4.5 MAL ATTRIBUTES

4.5.1 Blob

The Blob structure shall be used to hold binary data. It shall be a variable-length Octet array and the maximum length shall depend on the selected encoding.

NOTE – The distinction between this type and a list of octet Attributes is that this type may allow language mappings and encodings to use more efficient or appropriate representations.

Name	Blob
Extends	Attribute
Short Form Part	1

4.5.2 Boolean

The Boolean structure shall be used to hold Boolean Attributes. Possible values are 'True' or 'False'.

Name	Boolean
Extends	Attribute
Short Form Part	2

4.5.3 Duration

The Duration structure shall be used to hold duration Attributes at nanoseconds resolution. It can be negative because it may be used to represent offsets. The duration shall support a range between -2^{63} nanoseconds and $2^{63}-1$ nanoseconds (to allow representation as a 64-bit signed integer).

Name	Duration
Extends	Attribute
Short Form Part	3

4.5.4 Float

The Float structure shall be used to hold floating point Attributes using the IEEE 754 32-bit range (reference [3]).

NOTE – Three special values exist for this type: POSITIVE_INFINITY, NEGATIVE_INFINITY, and NaN (meaning 'not a number').

Name	Float
Extends	Attribute
Short Form Part	4

4.5.5 Double

The Double structure shall be used to hold floating point Attributes using the IEEE 754 64-bit range (reference [3]).

NOTE – Three special values exist for this type: POSITIVE_INFINITY, NEGATIVE_INFINITY, and NaN.

Name	Double
Extends	Attribute
Short Form Part	5

4.5.6 Identifier

The Identifier structure shall be used to hold an identifier and can be used for indexing. It is a variable-length Unicode string and the maximum length shall depend on the selected encoding. For some encoding/decoding bindings, the use of a numeric value might be appropriate for this Attribute, for example, via a dictionary (reference [4]).

Name	Identifier
Extends	Attribute
Short Form Part	6

4.5.7 Octet

The Octet structure shall be used to hold 8-bit signed Attributes. The permitted range is −128 to 127.

Name	Octet
Extends	Attribute
Short Form Part	7

4.5.8 UOctet

The UOctet structure shall be used to hold 8-bit unsigned Attributes. The permitted range is 0 to 255.

Name	UOctet
Extends	Attribute
Short Form Part	8

4.5.9 Short

The Short structure shall be used to hold 16-bit signed Attributes. The permitted range is −32768 to 32767.

Name	Short
Extends	Attribute
Short Form Part	9

4.5.10 UShort

The UShort structure shall be used to hold 16-bit unsigned Attributes. The permitted range is 0 to 65535.

Name	UShort
Extends	Attribute
Short Form Part	10

4.5.11 Integer

The Integer structure shall be used to hold 32-bit signed Attributes. The permitted range is −2147483648 to 2147483647.

Name	Integer
Extends	Attribute
Short Form Part	11

4.5.12 UInteger

The UInteger structure shall be used to hold 32-bit unsigned Attributes. The permitted range is 0 to 4294967295.

Name	UInteger
Extends	Attribute
Short Form Part	12

4.5.13 Long

The Long structure shall be used to hold 64-bit signed Attributes. The permitted range is −9223372036854775808 to 9223372036854775807.

Name	Long
Extends	Attribute
Short Form Part	13

4.5.14 ULong

The ULong structure shall be used to hold 64-bit unsigned Attributes. The permitted range is 0 to 18446744073709551615.

Name	ULong
Extends	Attribute
Short Form Part	14

4.5.15 String

The String structure shall be used to hold string Attributes. It is a variable-length Unicode string (reference [4]) and the maximum length shall depend on the selected encoding.

Name	String
Extends	Attribute
Short Form Part	15

4.5.16 Time

The Time structure shall be used to hold absolute time Attributes. It shall represent an absolute date and time to millisecond resolution. The range shall depend on the selected encoding.

Name	Time
Extends	Attribute
Short Form Part	16

4.5.17 FineTime

The FineTime structure shall be used to hold high-resolution absolute time Attributes. It shall represent an absolute date and time to nanosecond resolution. The range shall depend on the selected encoding.

Name	FineTime
Extends	Attribute
Short Form Part	17

4.5.18 URI

The URI structure shall be used to hold URI addresses. It shall be a variable-length Unicode string (references [4] and [5]) and the maximum length shall depend on the selected encoding.

Name	URI
Extends	Attribute
Short Form Part	18

4.5.19 ObjectRef

4.5.19.1 The ObjectRef structure shall be used to hold references to MO Objects. MO Objects are unambiguously identifiable by their unique Object Identity.

4.5.19.2 In order to reference the latest version of an MO Object, the version field of the ObjectRef shall be set to '0'.

4.5.19.3 The ObjectRef type shall be represented in specifications as ObjectRef<T> as presented in 4.2.7.

4.5.19.4 ObjectRef is a reserved type and no other area shall define a type with the same name.

Name	ObjectRef
Extends	Attribute
Short Form Part	19

4.6 MAL DATA STRUCTURES

4.6.1 InteractionType ENUMERATION

InteractionType is an enumeration that shall be used to hold the possible Interaction Pattern types.

Name	InteractionType	
Short Form Part	101	
Enumeration Value	Numeric Value	Comment
SEND	1	Used for SEND interactions.
SUBMIT	2	Used for SUBMIT interactions.
REQUEST	3	Used for REQUEST interactions.
INVOKE	4	Used for INVOKE interactions.
PROGRESS	5	Used for PROGRESS interactions.
PUBSUB	6	Used for PUBLISH-SUBSCRIBE interactions.

4.6.2 SessionType ENUMERATION

SessionType is an enumeration that shall be used to hold the session types. This facilitates the use of different Sessions in out-of-band agreements.

Name	SessionType	
Short Form Part	102	
Enumeration Value	Numeric Value	Comment
LIVE	1	Used for Live sessions.
SIMULATION	2	Used for Simulation sessions.
REPLAY	3	Used for Replay sessions.

4.6.3 QoSLevel ENUMERATION

QoSLevel is an enumeration that shall be used to hold the possible QoS levels. This facilitates the use of different QoS in out-of-band agreements.

Name	QoSLevel	
Short Form Part	103	
Enumeration Value	Numeric Value	Comment
BESTEFFORT	1	Used for Best Effort QoS Level.
ASSURED	2	Used for Assured QoS Level.
QUEUED	3	Used for Queued QoS Level.
TIMELY	4	Used for Timely QoS Level.

4.6.4 AttributeType ENUMERATION

AttributeType is an enumeration that shall be used to hold the defined MAL Attribute types.

Name	AttributeType	
Short Form Part	104	
Enumeration Value	Numerical Value	Comment
BLOB	1	Blob type.
BOOLEAN	2	Boolean type.
DURATION	3	Duration type.
FLOAT	4	Float type.
DOUBLE	5	Double type.
IDENTIFIER	6	Identifier type.
OCTET	7	Octet type.
UOCTET	8	UOctet type.
SHORT	9	Short type.
USHORT	10	UShort type.
INTEGER	11	Integer type.
UINTEGER	12	UInteger type.
LONG	13	Long type.
ULONG	14	ULong type.
STRING	15	String type.
TIME	16	Time type.
FINETIME	17	FineTime type.
URI	18	URI type.
OBJECTREF	19	ObjectRef type.

4.6.5 MOArea ENUMERATION

MOArea is an enumeration that shall be used to hold the known existing area numbers in use.

NOTE – The table follows the corresponding MO Area numbers available in the SANA registry (reference [7]) at time of publication of this document issue.

Name	MOArea	
Short Form Part	105	
Enumeration Value	Numeric Value	Comment
MAL	1	The MAL area number.
COM	2	The COM area number. The COM is deprecated; therefore this area number is reserved for backward compatibility.
COMMON	3	The Common area number.
MC	4	The Monitor and Control area number.
MPS	5	The Mission Planning and Scheduling area number.
SM	7	The Software Management area number.
MDPD	9	The Mission Data Product Distribution area number.

4.6.6 Subscription

The Subscription structure shall be used when subscribing for updates using the PUBSUB Interaction Pattern. It shall contain a single identifier that identifies the subscription being defined and a set of entities being requested.

Name	Subscription		
Extends	Composite		
Short Form Part	1001		
Field	Type	Nullable	Comment
subscriptionId	Identifier	No	The identifier of this subscription.
domain	List<Identifier>	Yes	Optional domain identifier. If NULL, the subscription shall match with any domain.
selectedKeys	List<Identifier>	Yes	The list of names of the selected Subscription Keys to be transmitted to the consumer. The Subscription Keys that are not in this list will be removed. If NULL, then all Subscription Keys will be transmitted.
filters	List<SubscriptionFilter>	Yes	The list of filters for this subscription. The list of filters must be ANDed together. If NULL, the subscription will not filter specific keys.

4.6.7 SubscriptionFilter

The SubscriptionFilter structure shall be used when subscribing for updates using the PUBSUB Interaction Pattern. It shall contain a single identifier that identifies the Subscription Key name and the set of values to be registered for the defined key name.

Name	SubscriptionFilter		
Extends	Composite		
Short Form Part	1002		
Field	Type	Nullable	Comment
name	Identifier	No	The identifier name of the key.
values	List<Attribute>	No	The list of values that are being subscribed for this key. These shall be ORed together.

4.6.8 UpdateHeader

The UpdateHeader structure shall be used by updates using the PUBSUB Interaction Pattern. It shall hold information that identifies a single update.

Name	UpdateHeader		
Extends	Composite		
Short Form Part	1003		
Field	Type	Nullable	Comment
source	Identifier	Yes	The source of the update, usually a PUBSUB provider.
domain	List<Identifier>	Yes	The domain of this update. The individual domain identifier parts shall not be set as the wildcard character '*'.
keyValues	List<NullableAttribute>	Yes	The values for the PUBSUB keys. The values shall be ordered according to the defined keys if the consumer subscription did not enable trimming.

4.6.9 IdBooleanPair

IdBooleanPair shall be a simple pair type of an identifier and Boolean value.

Name	IdBooleanPair		
Extends	Composite		
Short Form Part	1004		
Field	Type	Nullable	Comment
id	Identifier	No	The identifier value.
value	Boolean	Yes	The Boolean value.

4.6.10 Pair

Pair shall be a simple Composite structure for holding pairs.

NOTE – The pairs can be user-defined Attributes.

Name	Pair		
Extends	Composite		
Short Form Part	1005		
Field	Type	Nullable	Comment
first	Attribute	Yes	The Attribute value for the first Element of this pair.
second	Attribute	Yes	The Attribute value for the second Element of this pair.

4.6.11 NamedValue

The NamedValue structure shall represent a simple pair type of an identifier and abstract Attribute value.

Name	NamedValue		
Extends	Composite		
Short Form Part	1006		
Field	Type	Nullable	Comment
name	Identifier	No	The Identifier value.
value	Attribute	Yes	The Attribute value.

4.6.12 File

The File structure represents a file and shall be used to hold details about a file. It may also, optionally, hold a BLOB of the file data. The file type shall be denoted using the internet MIME media types.

NOTE – The list of official MIME types is held in reference [2].

Name	File		
Extends	Composite		
Short Form Part	1007		
Field	Type	Nullable	Comment
name	String	No	The file name.
mimeType	String	Yes	The MIME type of the file, NULL if not known.
creationDate	Time	Yes	The creation timestamp of the file, NULL if not known.
modificationDate	Time	Yes	The last modification timestamp of the file, NULL if not known.
size	ULong	Yes	The size of the file, NULL if not known.
content	Blob	Yes	The contents of the file, NULL if not supplied.
metaData	List<NamedValue>	Yes	A list of extra metadata for the file.

4.6.13 ObjectIdentity

The ObjectIdentity structure shall represent the Object Identity of an MO Object.

Name	ObjectIdentity		
Extends	Composite		
Short Form Part	1008		
Field	Type	Nullable	Comment
domain	List<Identifier>	No	The domain of the MO Object being referenced.
key	Identifier	No	The key of the MO Object being referenced.
version	UInteger	No	The version of the MO Object being referenced.

4.6.14 ServiceId

The ServiceId structure shall represent a specific service in MO.

Name	ServiceId		
Extends	Composite		
Short Form Part	1009		
Field	Type	Nullable	Comment
keyArea	UShort	No	The area of this service taken from the numeric Area identifier of the service specification.
keyService	UShort	No	The service taken from the numeric Service identifier of the service specification.
keyAreaVersion	UOctet	No	The Area Version of the service.

4.6.15 NullableAttribute

NullableAttribute structure shall represent an Attribute that can be nullable.

Name	NullableAttribute		
Extends	Composite		
Short Form Part	1010		
Field	Type	Nullable	Comment
value	Attribute	Yes	The value of the nullable attribute.

5 MO ERRORS

5.1 OVERVIEW

An error is typically defined as an incorrect behaviour or as something that is not correct. An MO Error is expected to occur when something goes wrong in the system or when an interaction is not completed successfully.

An example of a hypothetical MO Error that might occur in a system could be encryption failure during the encoding process of a message. Another example of a hypothetical MO Error that might occur when an interaction is not completed successfully could be a request to a service to get a parameter value for a parameter that does not exist.

MO Errors are defined at Area-level and can then be used by service operations. The specification of each operation explicitly includes the set of MO Errors to be used and their respective reason why they occur. MO Errors are composed of an Error Number, an Error Name, and an Error Description.

5.2 REQUIREMENTS

5.2.1 MO Errors shall be specified at Area-level.

5.2.2 Each service operation shall define the list of MO Errors that it uses and specify when they are expected to occur.

5.2.3 Each MO Error shall have an Error Number, Error Name, and Error Description.

5.2.4 The Error Number shall be within the valid range of the UInteger type.

5.2.5 The Error Number of the MAL area shall be in a range defined between 65536 and 65555 inclusive.

5.2.6 The Error Number in areas other than MAL shall be in the range defined between 1 and 65535 inclusive.

5.3 MAL-SPECIFIC MO ERRORS

The list of MO Errors specific to the MAL are defined in table 5-1. The MO Errors defined in this list may be used, not only by the MAL, but also by the service operations defined in other MO Areas.

Table 5-1: MAL-Specific MO Errors

Error Number	Error Name	Error Description
65536	Delivery Failed	Confirmed communication error.
65537	Delivery Timedout	Unconfirmed communication error.
65538	Delivery Delayed	Message queued somewhere awaiting contact.
65539	Destination Unknown	Destination cannot be contacted.
65540	Destination Transient	Destination middleware reports destination application does not exist.
65541	Destination Lost	Destination lost halfway through conversation.
65542	Authentication Failed	A failure to authenticate the message correctly.
65543	Authorisation Fail	A failure in the MAL to authorize the message.
65544	Encryption Fail	A failure in the MAL to encrypt/decrypt the message.
65545	Unsupported Area	The destination does not support the selected area.
65546	Unsupported Area Version	The destination does not support the selected area version.
65547	Unsupported Service	The destination does not support the selected service.
65548	Unsupported Operation	The destination does not support the selected operation.
65549	Bad Encoding	The destination was unable to decode the message.
65550	Internal	An internal error has occurred.
65551	Unknown	Operation specific.
65552	Incorrect State	The destination was not in the correct state for the received message.
65553	Too Many	Maximum number of subscriptions or providers of a broker has been exceeded.
65554	Shutdown	The component is being shutdown.
65555	Transaction Timeout	The interaction exceeded a certain timeout duration.

NOTES

1 Only two errors are possible from the MAL and below with regards to authentication and authorization. They are concerned with message-level issues and do not cover login issues such as an incorrect username/password combination.

2 The authentication and authorization failure messages are very generic in nature; it is possible that a weakness in an underlying protocol or authentication mechanism could be exploited if too much information were returned about the specific nature of an authentication or authorization failure.

5.4 DISCUSSION

The mapping of MO Errors to a concrete programming language is not herein explicitly defined. This is a deliberate decision in order to enable a wide range of possible mappings.

For example, a low-level programming language might use a simple mapping of the Error Number directly to a static variable, while a high-level programming language might map the MO Errors into a programming language ‘exception’ (supported in modern programming languages such as Java, C#, or C++).

6 MAL XML SPECIFICATION

6.1 OVERVIEW

This specification defines a normative XML Schema Definition (XSD) for validating MO service specifications and the MAL XML specification. The use of XML for service specification provides a machine-readable format rather than the text-based document format (reference [6]).

The published XML Schema Definition (XSD) and the service specifications are held in an online SANA registry (reference [7]) located at:

http://sanaregistry.org/r/moschemas/

6.2 XML SCHEMA DEFINITION (XSD) FOR MO SERVICES

The XML Schema Definition (XSD) that is used to validate the actual XML service specifications is located at:

http://sanaregistry.org/r/moschemas/ServiceSchema-vBBB.xsd

where the 'BBB' part is replaced with the issue number of the corresponding document.

6.3 MAL XML

6.3.1 The normative XML for the MAL specification, validated against the XML Schema Definition (XSD), is located at:

http://sanaregistry.org/r/moschemas/areaAAA-vBBB-MAL.xml

where the 'AAA' part is replaced with the area number ('001' for MAL) and the 'BBB' part is replaced with the area version that matches the issue number of the corresponding document.

6.3.2 The latest version of the MAL specification is directly available from the address:

http://sanaregistry.org/r/moschemas/ServiceDefMAL.xml

7 MO IN A CONCRETE DEPLOYMENT

7.1 OVERVIEW

As described in the introduction, the MAL is one of the composing parts of the MO layered service-oriented architecture. Any meaningful deployment of MO services requires the presence of the service adaption layer, the MAL, and the Transport Layer.

7.2 REQUIREMENTS

To achieve end-to-end interoperability between two Agencies using the MO services Architecture, an out-of-band agreement needs to be defined. This out-of-band agreement is expected to contain the following information:

a) the list of services deployed on the provider and its respective URIs; this list can be reduced to one single URI when a dedicated service for discovery is used (see reference [8]);

b) the transport mapping to be used (e.g., maltcp, malspp) and its specific mapping configuration parameters (if any);

c) the encoding to be used (e.g., Binary, Split Binary, XML);

d) security:

   1) mechanism for Access Control;

   2) usage of the ‘Authentication Id’ field of the MAL header messages (if any).

e) if the MAL header fields ‘From’ and ‘To’ are not URI-based, then their resolution from text to concrete URIs needs to be defined;

f) usage of supplementary MAL header fields (if required), and their respective meanings.

ANNEX A

PROTOCOL IMPLEMENTATION CONFORMANCE STATEMENT PROFORMA

(NORMATIVE)

A1 INTRODUCTION

A1.1 OVERVIEW

This annex provides the Protocol Implementation Conformance Statement (PICS) Requirements List (RL) for an implementation of the MO MAL Standard. The PICS for an implementation is generated by completing the RL in accordance with the instructions below. An implementation claiming conformance must satisfy the mandatory requirements referenced in the RL.

An implementation's completed RL is called the PICS. The PICS states which protocol features have been implemented. The following entities can use the PICS:

- the protocol implementer, as a checklist to reduce the risk of failure to conform to the standard through oversight;
- the supplier and acquirer or potential acquirer of the implementation, as a detailed indication of the capabilities of the implementation, stated relative to the common basis for understanding provided by the standard PICS proforma;
- the user or potential user of the implementation, as a basis for initially checking the possibility of interworking with another implementation (while interworking can never be guaranteed, failure to interwork can often be predicted from incompatible PICSes);
- a protocol tester, as the basis for selecting appropriate tests against which to assess the claim for conformance of the implementation.

A1.2 NOTATION

A1.2.1 Status Column Symbols

The following are used in the RL to indicate the status of features:

Symbol	Meaning
M	Mandatory
O	Optional

A1.2.2 Support Column Symbols

The support of every item as claimed by the implementer is stated by entering the appropriate answer (Y, N, or N/A) in the support column.

Symbol	Meaning
Y	Yes, supported by the implementation
N	No, not supported by the implementation
N/A	Not applicable

A2 GENERAL INFORMATION

A2.1 IDENTIFICATION OF PICS

Ref	Question	Response
1	Date of Statement (DD/MM/YYYY)	
2	CCSDS document number containing the PICS	
3	Date of CCSDS document containing the PICS	

A2.2 IDENTIFICATION OF IMPLEMENTATION UNDER TEST (IUT)

Ref	Question	Response
1	Implementation name	
2	Implementation version	
3	Machine name	
4	Machine version	
5	Operating System name	
6	Operating System version	
7	Special Configuration	
8	Other Information	

A2.3 USER IDENTIFICATION

Supplier	
Contact Point for Queries	
Implementation name(s) and Versions	
Other Information Necessary for full identification, for example, name(s) and version(s) for machines and/or operating systems; System Name(s)	

A2.4 INSTRUCTIONS FOR COMPLETING THE RL

An implementer shows the extent of compliance to the protocol by completing the RL; the resulting completed RL is called a PICS.

A3 ABSTRACT SERVICE SPECIFICATIONS PICS

There are six separate Interaction Patterns in the MAL. All Interaction Patterns are mandatory and must be implemented according to the specifications defined in this Recommended Standard. This section must be completed.

Ref	Interaction Pattern	Optional/Mandatory	Support
1	SEND	Mandatory	
2	SUBMIT	Mandatory	
3	REQUEST	Mandatory	
4	INVOKE	Mandatory	
5	PROGRESS	Mandatory	
6	PUBSUB	Mandatory	

The Access Control Interface includes the CHECK Interaction Pattern. The CHECK Interaction Pattern is mandatory and must be implemented according to the specifications defined in this Recommended Standard. This section must be completed.

Ref	Access Control Interface Pattern	Optional/Mandatory	Support
1	CHECK	Mandatory	

A4 MAL DATA TYPE SPECIFICATION PICS

All Fundamental Data Types are mandatory and must be implemented according to the specifications defined in this Recommended Standard. This includes all defined Attributes and data structures in this Recommended Standard. This section must be completed.

Ref	Fundamentals	Optional/Mandatory	Support
1	Element	Mandatory	
2	Attribute	Mandatory	
3	Composite	Mandatory	
4	Object	Mandatory	

A5 MAL ERRORS PICS

All MAL errors are mandatory and must be implemented according to the specifications defined in this Recommended Standard. This section must be completed.

Ref	Error	Optional/Mandatory	Support
1	Delivery Failed	Mandatory	
2	Delivery Timedout	Mandatory	
3	Delivery Delayed	Mandatory	
4	Destination Unknown	Mandatory	
5	Destination Transient	Mandatory	
6	Destination Lost	Mandatory	
7	Authentication Failed	Mandatory	
8	Authorisation Fail	Mandatory	
9	Encryption Fail	Mandatory	
10	Unsupported Area	Mandatory	
11	Unsupported Area Version	Mandatory	
12	Unsupported Service	Mandatory	
13	Unsupported Operation	Mandatory	
14	Bad Encoding	Mandatory	
15	Internal	Mandatory	
16	Unknown	Mandatory	
17	Incorrect State	Mandatory	
18	Too Many	Mandatory	
19	Shutdown	Mandatory	
20	Transaction Timeout	Optional	

ANNEX B

SECURITY, SANA, AND PATENT CONSIDERATIONS

(INFORMATIVE)

B1 SECURITY CONSIDERATIONS

B1.1 ANALYSIS OF SECURITY CONSIDERATIONS

This annex presents the results of an analysis of security considerations applied to the technologies specified in this Recommended Standard.

B1.2 CONSEQUENCES OF NOT APPLYING SECURITY TO THE TECHNOLOGY

The current specification has two aspects: the design time aspect and the runtime aspect. For the design time aspect, this specification provides an abstract service and data specification language. This language is used in other Blue Book specifications to specify application-level services and data models in the domain of MO. Hence at the level of this Recommended Standard, no generally valid assumptions can be made about the nature of the services and data that will be specified in terms of MAL. Accordingly, no generally valid consequences can be assumed as a generic statement with regard to not applying ‘security to the technology’. Same holds for the runtime aspects of MAL. For runtime aspects, MAL specifies a generic Access Control Interface and delegates the responsibilities of implementing the adequate security controls, which are deployment specific, to the implementation of the deployment-specific Access Control component. That said, regardless of the nature of the services that will be specified in the future, using the language provided by this specification, if no security requirements are applied to the Application Layer, the Access Control component, and the Transport Layer (all out of the scope of this specification) in a concrete implementation of this specification for a given deployment configuration, the confidentiality, integrity, and availability of the respected services and data could be compromised.

B1.3 POTENTIAL THREATS AND ATTACK SCENARIOS

Potential threats or attack scenarios include, but are not limited to, (a) unauthorized access to the programs/processes that generate and interpret the messages, and (b) unauthorized access to the messages during transmission between exchange partners. Protection from unauthorized access during transmission is especially important if the mission utilizes open ground networks such as the Internet to provide ground station connectivity for the exchange of data formatted in compliance with this Recommended Standard. It is strongly recommended that potential threats or attack scenarios applicable to the systems and networks on which this Recommended Standard is implemented be addressed by the management of those systems and networks.

B1.4 DATA PRIVACY

Privacy of data formatted in compliance with the specifications of this Recommended Standard should be assured by the systems and networks on which this Recommended Standard is implemented.

B1.5 DATA INTEGRITY

Integrity of data formatted in compliance with the specifications of this Recommended Standard should be assured by the systems and networks on which this Recommended Standard is implemented.

B1.6 AUTHENTICATION OF COMMUNICATING ENTITIES

Authentication of communicating entities involved in the transport of data which complies with the specifications of this Recommended Standard should be provided by the systems and networks on which this Recommended Standard is implemented.

B1.7 DATA TRANSFER BETWEEN COMMUNICATING ENTITIES

The transfer of data formatted in compliance with this Recommended Standard between communicating entities should be accomplished via secure mechanisms approved by the IT Security functionaries of exchange participants.

B1.8 CONTROL OF ACCESS TO RESOURCES

Control of access to resources should be managed by the systems upon which originator formatting and recipient processing are performed.

B1.9 AUDITING OF RESOURCE USAGE

Auditing of resource usage should be handled by the management of systems and networks on which this Recommended Standard is implemented.

B1.10 UNAUTHORIZED ACCESS

Unauthorized access to the programs/processes that generate and interpret the messages should be prohibited in order to minimize potential threats and attack scenarios.

B1.11 DATA SECURITY IMPLEMENTATION SPECIFICS

Specific information-security interoperability provisions that may apply between agencies and other independent users involved in an exchange of data formatted in compliance with this Recommended Standard should be specified in an ICD.

B2 SANA CONSIDERATIONS

The recommendations of this document request SANA to create a registry named 'Mission Operations Service XML' that consists of a set of XML schemas and XML service specifications.

The registration rule for change to this registry requires an engineering review by a designated expert. The expert shall be assigned by the working group chair, or in absence, the Area Director.

Additionally, the recommendations of this document request SANA to create a registry named 'Mission Operations MO Areas' that consists of a set of entries containing the MO Area, MO Area Names, and its corresponding MO Area Numbers and a comment, as follows:

Area	Area Name	Area Number	Comment
MAL	Message Abstraction Layer	1	The MAL area number.
COM	Common Object Model services	2	The COM area number. The COM is deprecated; therefore this area number is reserved for backward compatibility.
COMMON	Common services	3	The Common area number.
MC	Monitor and Control services	4	The Monitor and Control area number.
MPS	Mission Planning and Scheduling services	5	The Mission Planning and Scheduling area number.
SM	Software Management services	7	The Software Management area number.
MDPD	Mission Data Product Distribution services	9	The Mission Data Product Distribution area number.

B3 PATENT CONSIDERATIONS

The recommendations of this document have no patent issues.

ANNEX C

ABBREVIATIONS AND ACRONYMS

(INFORMATIVE)

ACK	acknowledgment
API	application programming interface
APID	application identifier
ASCII	American Standard Code for Information Interchange
BLOB	binary large object
CCSDS	Consultative Committee for Space Data Standards
DNS	Domain Name System
ICD	interface control document
IEEE	Institute of Electrical and Electronics Engineers
IT	information technology
IUT	implementation under test
MAL	Message Abstraction Layer
MO	Mission Operations
NaN	not a number
PICS	protocol implementation conformance statement
RL	requirements list
PICS	Protocol Implementation Conformance Statement
PUBSUB	Publish-Subscribe
QoS	quality of service
RL	requirements list
RPC	remote procedure call
SANA	Space Assigned Numbers Authority
SM&C	Spacecraft Monitor & Control
URI	Universal Resource Identifier
XSD	XML Schema Definition
XML	Extensible Markup Language

ANNEX D

INFORMATIVE REFERENCES

(INFORMATIVE)

[D1] *Mission Operations Services Concept*. Report Concerning Space Data System Standards, CCSDS 520.0-G-3. Green Book. Issue 3. Washington, D.C.: CCSDS, December 2010.

NOTE – Normative references are listed in 1.5.