The Consultative Committee for Space Data Systems

Recommendation for Space Data System Standards

MISSION OPERATIONS MESSAGE ABSTRACTION LAYER

RECOMMENDED STANDARD

CCSDS 521.0-B-1

BLUE BOOK

September 2010

The Consultative Committee for Space Data Systems

Recommendation for Space Data System Standards

MISSION OPERATIONS MESSAGE ABSTRACTION LAYER

RECOMMENDED STANDARD

CCSDS 521.0-B-1

BLUE BOOK

September 2010

AUTHORITY

Issue:	Recommended Standard, Issue 1
Date:	September 2010
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 the *Procedures Manual for the Consultative Committee for Space Data Systems*, and the record of Agency participation in the authorization of this document can be obtained from the CCSDS Secretariat at the address below.

This document is published and maintained by:

CCSDS Secretariat
Space Communications and Navigation Office, 7L70
Space Operations Mission Directorate
NASA Headquarters
Washington, DC 20546-0001, USA

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

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 the *Procedures Manual for the Consultative Committee for Space Data Systems.* 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 addressed to the CCSDS Secretariat at the 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 e.V. (DLR)/Germany.
- European Space Agency (ESA)/Europe.
- Instituto Nacional de Pesquisas Espaciais (INPE)/Brazil.
- Japan Aerospace Exploration Agency (JAXA)/Japan.
- National Aeronautics and Space Administration (NASA)/USA.
- Russian Federal Space Agency (RFSA)/Russian Federation.
- UK Space Agency/United Kingdom.

Observer Agencies

- Austrian Space Agency (ASA)/Austria.
- Belgian Federal Science Policy Office (BFSPO)/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.
- Chinese Academy of Space Technology (CAST)/China.
- Commonwealth Scientific and Industrial Research Organization (CSIRO)/Australia.
- CSIR Satellite Applications Centre (CSIR)/Republic of South Africa.
- Danish National Space Center (DNSC)/Denmark.
- Departamento de Ciência e Tecnologia Aeroespacial (DCTA)/Brazil.
- 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.
- Indian Space Research Organization (ISRO)/India.
- Institute of Space Research (IKI)/Russian Federation.
- KFKI Research Institute for Particle & Nuclear Physics (KFKI)/Hungary.
- Korea Aerospace Research Institute (KARI)/Korea.
- Ministry of Communications (MOC)/Israel.
- 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.
- Scientific and Technological Research Council of Turkey (TUBITAK)/Turkey.
- Space and Upper Atmosphere Research Commission (SUPARCO)/Pakistan.
- Swedish Space Corporation (SSC)/Sweden.
- 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	Current issue

CONTENTS

CONTENTS (continued)

Section Page

Figure

CONTENTS (continued)

Figure Page

Table

CONTENTS (continued)

Table Page

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 [B1], Mission Operations Services Concept.

The MO MAL is a framework that provides generic service patterns to the Mission Operation services defined in reference [B1]. These Mission Operations 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 organised 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 MAL errors;

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

g) section 7 is 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.

Hardware Component: a complex physical entity (such as a spacecraft, a tracking system, or a control system) or an individual physical entity of a system (such as an instrument, a computer, or a piece of communications equipment). A Hardware Component may be composed from other Hardware Components. Each Hardware Component may host one or more Software Components. Each Hardware Component has one or more ports where connections to other Hardware Component are made. Any given Port on the Hardware Component may expose one or more 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 where 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.

Service System: the set of Hardware and Software Components used to implement a Service in a real system. Service Systems may be implemented using one or more Hardware and Software Components.

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 (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.

Protocol Stack: the stack of Protocol Layers required for communication.

Protocol Layer: the implementation of a specific Protocol. It provides a Protocol Service Access Point to layers above and uses the Protocol Service Access Point of the layer below.

Protocol Service Access Point (SAP): the point at which one layer's functions are provided to the layer above. A layer may provide protocol services to one or more higher layers and use the protocol services of one or more lower layers. A SAP defines unambiguously the interface for a protocol that may be used as part of a Service Interface Binding specification.

Protocol: the set of rules and formats (semantic and syntactic) used to determine the communication behaviour of a Protocol Layer in the performance of the layer functions. The state machines that operate and the protocol data units that are exchanged specify a protocol.

Service directory: an entity that provides publish and lookup facilities to service providers and consumers.

NOTE – Strictly speaking, a directory is not required if a well-known service is to be used; however, in most circumstances a directory provides required flexibility in the location of services. Service location can be statically configured, dynamically discovered through a service directory, or a combination of the two; this is an implementation choice. The service directory is itself, by definition, a service.

1.5 NOMENCLATURE

The following conventions apply throughout 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.

1.6 REFERENCES

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

[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.

NOTE – Informative references are listed in annex B.

2 OVERVIEW

2.1 GENERAL

The MO service specifications detail a standard set of services. These 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. There are three abstract interfaces defined in the MAL specification: the first is the abstract interface of the MAL that is presented to the higher layers, the second is the abstract interface of the Access Control component, and the third is the abstract interface that a Transport layer must provide to the MAL.

The MO service specifications and the MAL are abstract in their definition; they do not contain any specific information on how to implement them for a particular programming language or transport encoding. Moving from the abstract to the implemented system, two other specifications are needed. One is the Language Mapping that states how the abstract MAL and MO service specifications are to be realised in some specific language: this defines the API in that language. The second is the transport 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. For further information about the MO Concept see reference [B1], and for the Reference Model see 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 (giving 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 2-1) 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:

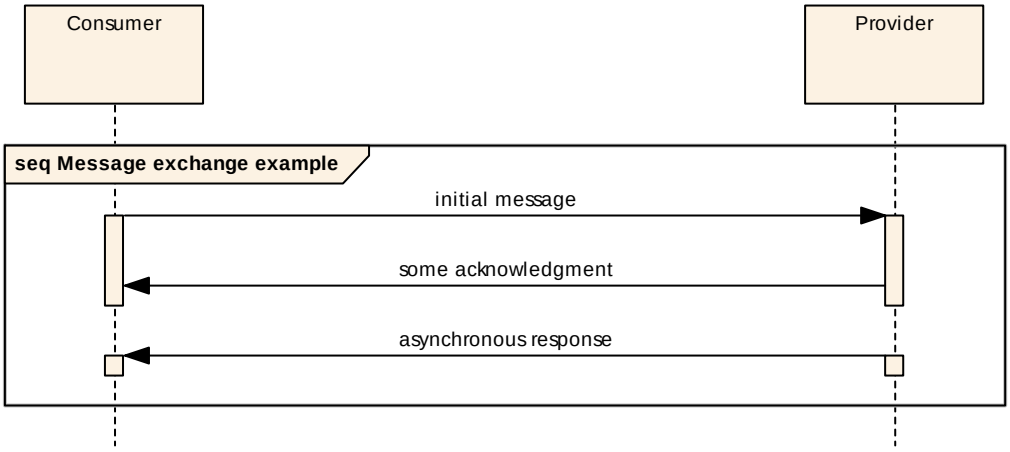

Figure 2-1: Message Exchange Sequence Example

Because the pattern shows the response, the consumer can expect it, but because the bars are not connected it 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 that 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	Body Type
<<Message direction>>	<<Message name>>	<<Message type>>
…	…	…

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.

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; Indications are transitions that are triggered by an external activity. Messages are the entities that are transferred between the two components to progress the interaction:

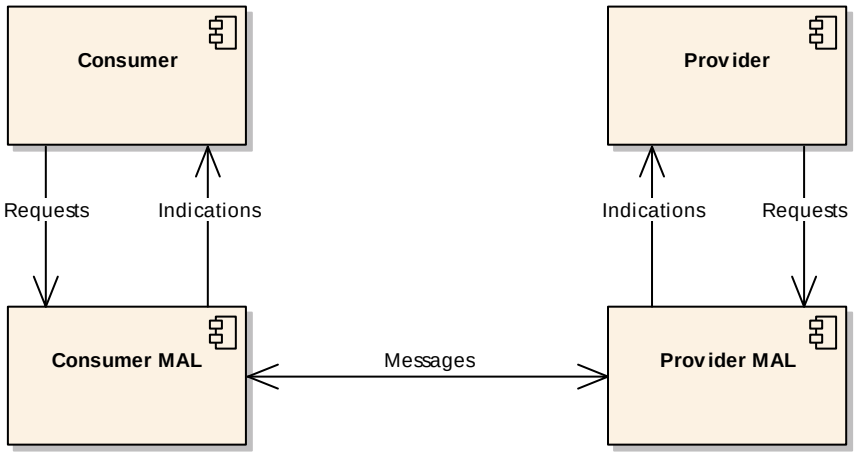

Figure 2-2: Request, Indication, and Message relationship

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 diagram:

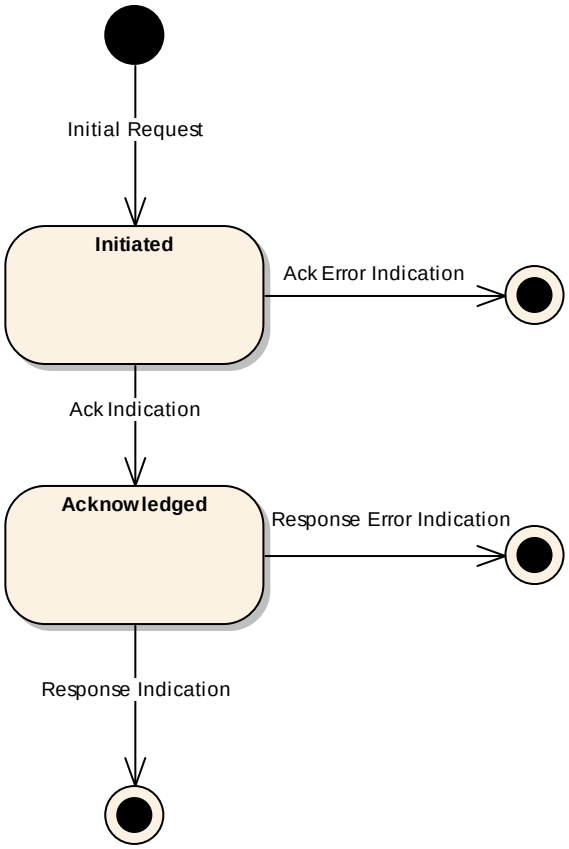

Figure 2-3: Consumer State Diagram Example

The diagrams define the allowed states and transitions between those states. The states are those of the MAL of the relevant component and apply to a single instance of a pattern; for example, the above diagram 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. The state also only applies to a single instance of a pattern; as there can be many interactions concurrently active with one MAL, it needs to be able to track the state of each interaction instance independently.

Figure 2-3 shows the state diagram for Consumer component from the example in figure 2-1, 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 are 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 what the effects of the reception of a Request by a MAL or an Indication by an Application are. Usually for a Request is it the transmission of a Message to the destination and 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 4.4.5.

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 structure. It is important to note that such a type of structure is completely dependent on the chosen protocol.

For the example decomposition each message is split into up to three parts (figure 2-4), a standard message header, possibly a standard pattern body, and, finally, possibly a service-specific body:

Figure 2-4: Message Decomposition Key

Each message, in the examples, is split into the relevant fields; the example value is shown in the box, and the field name is given above (figure 2-5):

Message Header															
URI From	*Auth Id*	*URI To*	*Timestamp*	*QoS*	*Priority*	*Domain*	*Zone*	*Session*	*Interaction*	*Stage*	*Trans Id*	*Area*	*Service*	*Operation*	*Version*	*Is Error*
Provider	SC X	Broker		BEST	1	A.B.C	GROUND	LIVE	PUBSUB	3		Example	Example	testPubSub	1	FALSE

Figure 2-5: Message Header Decomposition Example

When more complex structures are being shown (figure 2-6), a single field can be part of a more complex structure which is itself part of another composite structure:

UpdateList																	
List count	*Test Notify*								*Test Notify*								
	Time stamp	*Source URI*	*Update Type*	*IdentifierList*			*Time*	*Value*	*Time stamp*	*Source URI*	*Update Type*	*IdentifierList*			*Time*	*Value*
				List Count	*Id*	*Id*						*List Count*	*Id*	*Id*		
2		SC X	Update	2	A	B	Today, 09:30	1234		SC X	Update	2	A	C	Today, 09:30	8888

Figure 2-6: Message Body Decomposition Example

In the example above the ‘Update Type’ field is part of a TestNotify structure which is itself part of the overall UpdateList structure.

The above example also demonstrates the difference between a pattern body and a service-specific body. The UpdateList is the pattern body, but it contains a part that is specific to the service in question, the TestNotify part; however, only the last two fields of the TestNotify are specific to it and the preceding fields are inherited from the Update structure (these therefore are marked as part of the standard pattern body).

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 ABSTRACT SERVICE SPECIFICATIONS

Each abstract service specification Blue Book is specified as an Area that consists of one or more services that are composed of a set of operations. Service Areas provide a simple way of grouping related Services together.

Each service specification includes an overview table that details the Area and Service Identifiers and then lists the operations of that service:

Table 2-3: Service Overview Table

Area Identifier	Service Identifier	Area Number	Service Number	Service Version
Interaction Pattern	Operation Identifier	Operation Number	Support in replay	Capability Set

The values of table 2-3 are used to populate the [Area Identifier][Service Identifier][Operation Identifier][Service 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. These 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 Service Version is a number that is used to differentiate between issues of a service specification. Initially it will be set to the number ‘1’ and future updates to the service specification will increase that number.

Support in replay is used to denote whether a specific operation is allowed to be used in a Replay session and is either of the value ‘Yes’ or ‘No’. For example, only operations that do not modify history are usually supported in a replay session; operations such as setting a parameter would not be supported as they modify history.

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

3 ABSTRACT SERVICE SPECIFICATIONS

3.1 GENERAL

There are three 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 3.5.

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; however, the interaction with that component is part of the MAL standard. It is detailed in 3.6.

The third is the abstract interface that a Transport layer provides to the MAL. It specifies what facilities must be made available to a compliant MAL and also the required behaviour of the Transport. It is detailed in 3.7.

General constraints that apply to all patterns are defined in 3.2, 3.3, and 3.4.

3.2 TRANSACTION HANDLING

a) The message header shall contain a transaction identifier field providing a mechanism for the originating MAL of a message exchange to uniquely identify the response, or set of responses, to a message.

b) Each instance of a pattern shall be considered a single transaction.

c) The transaction identifier shall be:

    1) used by the originating MAL to identify messages in a particular instance of an interaction pattern, and

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

d) The originating MAL shall ensure that the combination of the following Message Header fields form a unique index: pattern type, transaction identifier, the source address, the session, the domain, the network, the service area identifier , the service identifier , and operation identifiers,

3.3 STATE TRANSITIONS

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

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

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

d) 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.

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

f) 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.

3.4 MESSAGE HEADER FIELD VALUES

a) The header fields, with the values specified in table 3-1, shall be used for all interaction patterns.

b) For the fields which values are left blank in table 3-1, the value shall be defined in the relevant Request and Indication section.

Table 3-1: MAL Message Header Fields

Field	Value
URI From	
Authentication Id	
URI To	
Timestamp	Message generation timestamp
QoSlevel	The QoS level of the message
Priority	The QoS priority of the message
Domain	Domain of the message
Network Zone	Network zone of the message
Session	Session of the message
Session Name	Session Identifier for Simulation and Replay sessions
Interaction Type	
Interaction Stage	
Transaction Id	
Service Area	Area Identifier

Field	Value
Service	Service Identifier
Operation	Operation Identifier
Service version	Service Version number
Is Error Message	FALSE

3.5 MAL SERVICE INTERFACE

3.5.1 SEND INTERACTION PATTERN

3.5.1.1 Overview

The SEND pattern is the most basic interaction (figure 3-1); it is a single message from the consumer to the provider. No acknowledgement is sent by the provider.

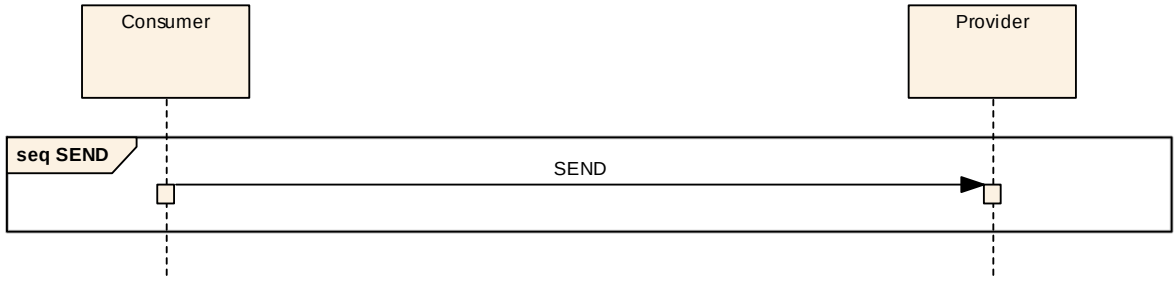

Figure 3-1: SEND Interaction Pattern Message Sequence

3.5.1.2 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. Because there is no message 'conversation' implied with a simple SEND, there is no requirement for a transaction identifier in the message. However, one may be specified. No return message is sent from the provider to the consumer, so the consumer has no indication the provider has received the message.

3.5.1.3 Usage

The SEND pattern is expected to be used for non-critical messages where the possible loss of one or more of these messages is not considered critical. An example would be regular heartbeat-type messages.

3.5.1.4 Error Handling

Errors (see 4.4.6) may be generated by the consumer service layers, but no error can be generated by the provider as the interaction pattern does not allow for the provider to return an error.

3.5.1.5 Operation Template

The SEND template only contains the operation name and the type of the structure that is sent as the message body:

Table 3-2: SEND Operation Template

Operation Identifier	<<Operation name>>	
Interaction Pattern	SEND	
Pattern Sequence	Message	Body Type
IN	SEND	<<Body type>>

3.5.1.6 Primitives

Table 3-3: SEND Primitive List

Primitive
SEND

3.5.1.7 State Charts

3.5.1.7.1 Consumer Side

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

3.5.1.7.2 Provider Side

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

3.5.1.8 Requests and Indications

3.5.1.8.1 SEND

3.5.1.8.1.1 Function

a) The **SEND Request** primitive shall be used by the consumer application to initiate a SEND Interaction.

b) The **SEND Indication** primitive shall be used by the provider MAL to deliver a **SEND Message** to a provider and initiate a SEND Interaction.

3.5.1.8.1.2 Semantics

SEND Request and **SEND Indication** shall provide parameters as follows:

(SEND Message Header, SEND Message Body, QoS properties)

3.5.1.8.1.3 When Generated

a) A **SEND Request** may be generated by the consumer application at any time.

b) 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.

3.5.1.8.1.4 Effect on Reception

a) 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.

b) The pattern shall end at this point for the consumer.

c) On reception of a **SEND Indication** a provider shall process the operation.

d) The pattern shall end at this point for the provider.

3.5.1.8.1.5 Message Header

a) For the **SEND Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-4.

b) 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
URI From	Consumer URI
Authentication Id	Consumer Authentication Identifier
URI To	Provider URI
Interaction Type	SEND
Interaction Stage	Not used
Transaction Id	Not used

3.5.1.9 Example

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

Operation Identifier	testSend	
Interaction Pattern	SEND	
Pattern Sequence	Message	Body Type
IN	SEND	TestBody

The TestBody structure is defined below:

Structure Name	TestBody	
Extends	Composite	
Short form	Example Only	
Field	Type	Comment
FirstItem	String	Example String item
SecondItem	Integer	Example Integer item

This corresponds to the following message:

Message Header																	*Test Body*	
URI From	*Auth Id*	*URI To*	*Timestamp*	*QoS*	*Priority*	*Domain*	*Zone*	*Session*	*Interaction*	*Stage*	*Trans Id*	*Area*	*Service*	*Operation*	*Version*	*Is Error*	*First Item*	*Second Item*
Consumer	Op A	Provider		BEST	1	A.B.C	GROUND	LIVE	SEND	1	1233	Example	Example	testSend	1	FALSE	Hello	1234

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.

3.5.2 SUBMIT INTERACTION PATTERN

3.5.2.1 Overview

The SUBMIT pattern is a simple confirmed message exchange pattern (figure 3-2). The consumer sends a message to a provider and the provider acknowledges it.

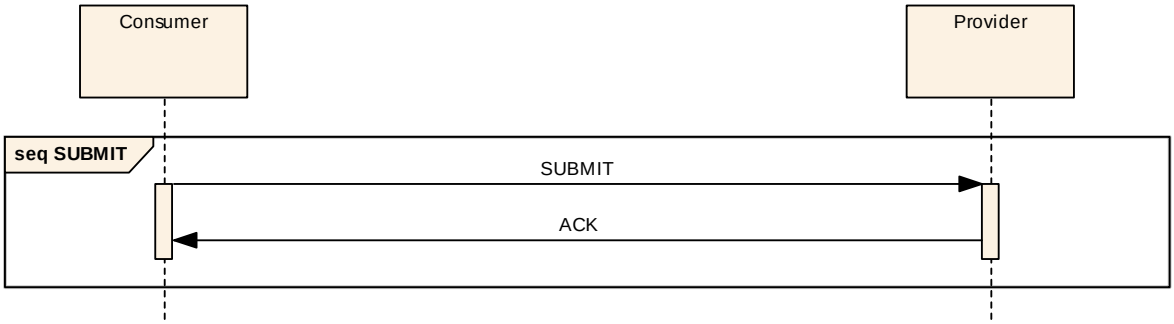

Figure 3-2: SUBMIT Interaction Pattern Message Sequence

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.

3.5.2.2 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.

3.5.2.3 Usage

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

3.5.2.4 Error Handling

a) The return acknowledgment shall be replaced with an error message (see 4.4.6) if an error occurs during the processing of the operation as shown in figure 3-3.

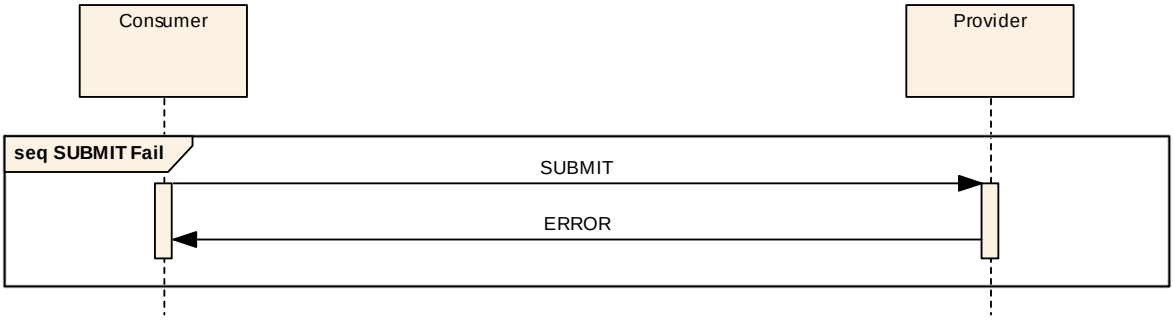

Figure 3-3: SUBMIT Interaction Pattern Error Sequence

b) Either the acknowledgment or the error message shall be returned but never both.

c) 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.

3.5.2.5 Operation Template

The SUBMIT template only contains the operation name and the type of the structure that is submitted as the Submit message body:

Table 3-5: SUBMIT Operation Template

Operation Identifier	<<Operation name>>	
Interaction Pattern	SUBMIT	
Pattern Sequence	Message	Body Type
IN	SUBMIT	<<Body type>>

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

NOTE – 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.

3.5.2.6 Primitives

Table 3-6: SUBMIT Primitive List

Primitive
SUBMIT
ACK
ERROR

3.5.2.7 State Charts

3.5.2.7.1 Consumer Side

Figure 3-4 shows the consumer side state chart for the SUBMIT pattern:

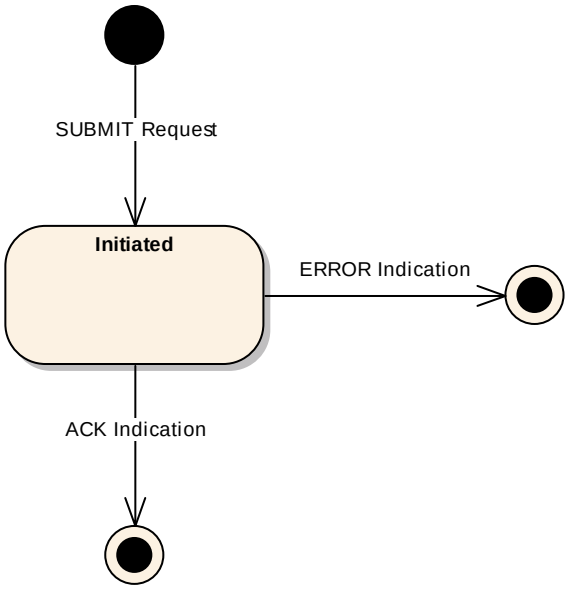

Figure 3-4: SUBMIT Consumer State Chart

a) The initial transition is triggered by the primitive **SUBMIT Request** and shall lead to the **Initiated State**.

b) There are two possible transitions from the **Initiated State**:

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

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

3.5.2.7.2 Provider Side

Figure 3-5 shows the provider side state chart for the SUBMIT pattern:

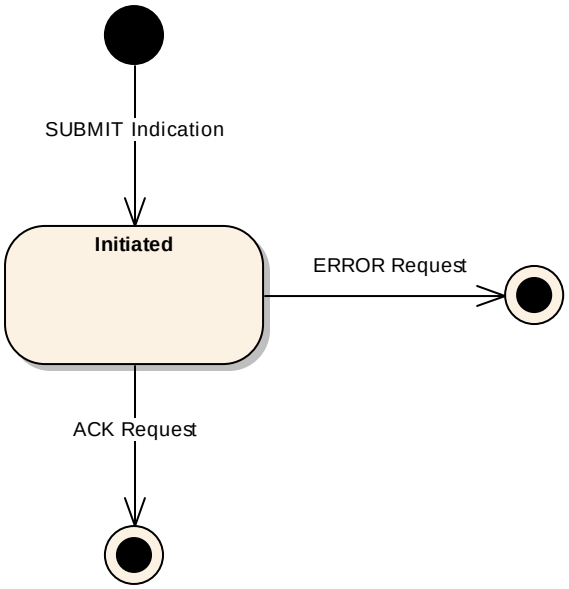

Figure 3-5: SUBMIT Provider State Chart

a) The initial transition is triggered by the primitive **SUBMIT Indication** and shall lead to the **Initiated State**.

b) There are two possible transitions from the **Initiated State**:

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

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

3.5.2.8 Requests and Indications

3.5.2.8.1 SUBMIT

3.5.2.8.1.1 Function

a) The **SUBMIT Request** primitive shall be used by the consumer application to initiate a SUBMIT Interaction.

b) The **SUBMIT Indication** primitive shall be used by the provider MAL to deliver a **SUBMIT Message** to a provider and initiate a SUBMIT Interaction.

3.5.2.8.1.2 Semantics

SUBMIT Request and **SUBMIT Indication** shall provide parameters as follows:

(SUBMIT Message Header, SUBMIT Message Body, QoS properties)

3.5.2.8.1.3 When Generated

a) A **SUBMIT Request** may be generated by the consumer application at any time.

b) 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.

3.5.2.8.1.4 Effect on Reception

a) 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.

b) The consumer MAL shall then enter the **Initiated State**.

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

d) The provider shall then start processing the operation at this point.

3.5.2.8.1.5 Message Header

a) For the **SUBMIT Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-7.

b) 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
URI From	Consumer URI
Authentication Id	Consumer Authentication Identifier
URI To	Provider URI
Interaction Type	SUBMIT
Interaction Stage	1
Transaction Id	Provided by consumer MAL

3.5.2.8.2 ACK

3.5.2.8.2.1 Function

a) The **ACK Request** primitive shall be used by the provider application to acknowledge a SUBMIT Interaction.

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

3.5.2.8.2.2 Semantics

ACK Request and **ACK Indication** shall provide parameters as follows:

(ACK Message Header, QoS properties)

3.5.2.8.2.3 When Generated

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

b) 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.

3.5.2.8.2.4 Effect on Reception

a) 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.

b) The pattern shall end at this point for the provider.

c) On reception of an **ACK Indication** a consumer shall end the interaction pattern with success.

3.5.2.8.2.5 Message Header

For the **ACK message** the message header fields shall be as defined in 3.4 except for the fields in table 3-8.

Table 3-8: Submit ACK Message Header Fields

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

3.5.2.8.3 ERROR

3.5.2.8.3.1 Function

a) The **ERROR Request** primitive shall be used by the provider application to end a SUBMIT Interaction with an error.

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

3.5.2.8.3.2 Semantics

ERROR Request and **ERROR Indication** shall provide parameters as follows:

(ERROR Message Header, StandardError, QoS properties)

3.5.2.8.3.3 When Generated

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

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

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

2) an error raised by the local communication layer.

3.5.2.8.3.4 Effect on Reception

a) 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.

b) The pattern shall end at this point for the provider.

c) On reception of an **ERROR Indication** a consumer shall end the interaction with an error.

3.5.2.8.3.5 Message Header

a) 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.

b) The contents of the StandardError structure shall immediately follow the message header.

3.5.2.9 Example

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

Operation Identifier	testSubmit	
Interaction Pattern	SUBMIT	
Pattern Sequence	Message	Body Type
IN	SUBMIT	TestBody

The TestBody structure is defined in 3.5.1.9. This corresponds to the following message being transmitted:

Message Header																	*Test Body*	
URI From	*Auth Id*	*URI To*	*Timestamp*	*QoS*	*Priority*	*Domain*	*Zone*	*Session*	*Interaction*	*Stage*	*Trans Id*	*Area*	*Service*	*Operation*	*Version*	*Is Error*	*First Item*	*Second Item*
Consumer	Op A	Provider		BEST	1	A.B.C	GROUND	LIVE	SUBMIT	1	1234	Example	Example	testSubmit	1	FALSE	Hello	1234

and corresponds to the following acknowledgement being returned:

Message Header																
URI From	*Auth Id*	*URI To*	*Timestamp*	*QoS*	*Priority*	*Domain*	*Zone*	*Session*	*Interaction*	*Stage*	*Trans Id*	*Area*	*Service*	*Operation*	*Version*	*Is Error*
Provider	SC X	Consumer		BEST	1	A.B.C	GROUND	LIVE	SUBMIT	2	1234	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.

3.5.3 REQUEST INTERACTION PATTERN

3.5.3.1 Overview

The REQUEST pattern extends the SUBMIT pattern by replacing the simple acknowledgement with a data response message (figure 3-6). No acknowledgement other than the data response is sent.

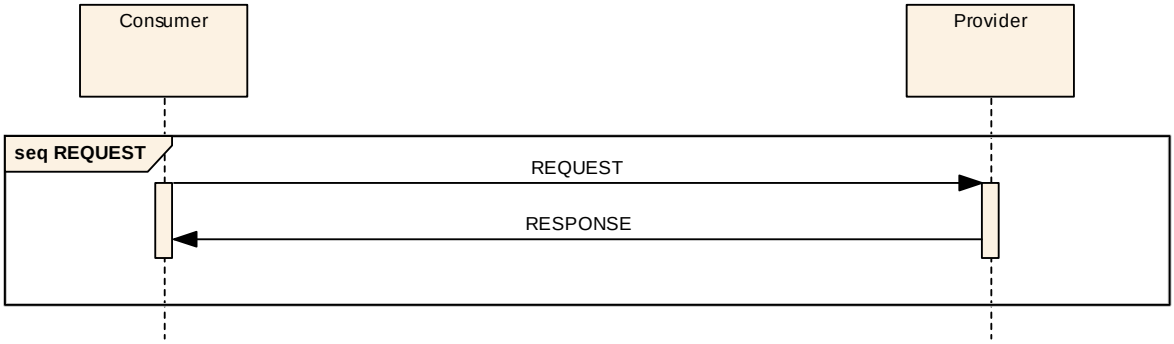

Figure 3-6: REQUEST Interaction Pattern Message Sequence

3.5.3.2 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 and only the return data response for this pattern means that it is primarily expected to be used for situations where the operation takes minimal time.

3.5.3.3 Usage

It is expected that the REQUEST pattern is to be used only for operations that complete 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.

3.5.3.4 Error Handling

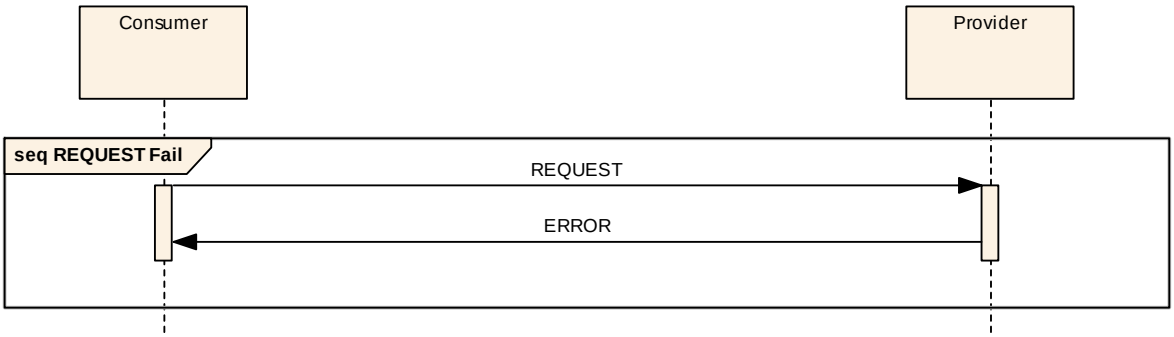

Figure 3-7: REQUEST Interaction Pattern Error Sequence

a) The data response shall be replaced with an error message (see 4.4.6) if an error occurs during the processing of the operation (figure 3-7).

b) Either the data response or the error message shall be returned but never both.

3.5.3.5 Operation Template

The REQUEST pattern template extends the SUBMIT template by adding the requirement for a return type:

Table 3-9: REQUEST Operation Template

Operation Identifier	<<Operation name>>	
Interaction Pattern	REQUEST	
Pattern Sequence	Message	Body Type
IN	REQUEST	<<Request type>>
OUT	RESPONSE	<<Response type>>

3.5.3.6 Primitives

Table 3-10: REQUEST Primitive List

Primitive
REQUEST
RESPONSE
ERROR

3.5.3.7 State Charts

3.5.3.7.1 Consumer Side

Figure 3-8 shows the consumer side state chart for the REQUEST pattern:

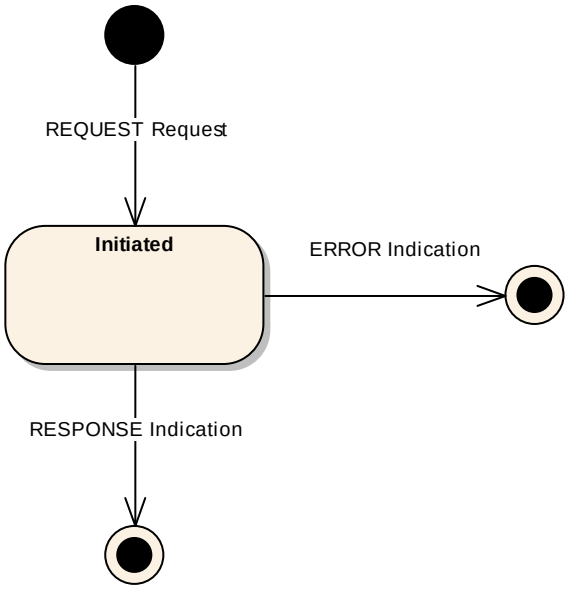

Figure 3-8: REQUEST Consumer State Chart

a) The initial transition is triggered by the primitive **REQUEST Request** and shall lead to the **Initiated State**.

b) There are two possible transitions from the **Initiated State**:

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

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

3.5.3.7.2 Provider Side

Figure 3-9 shows the provider side state chart for the REQUEST pattern:

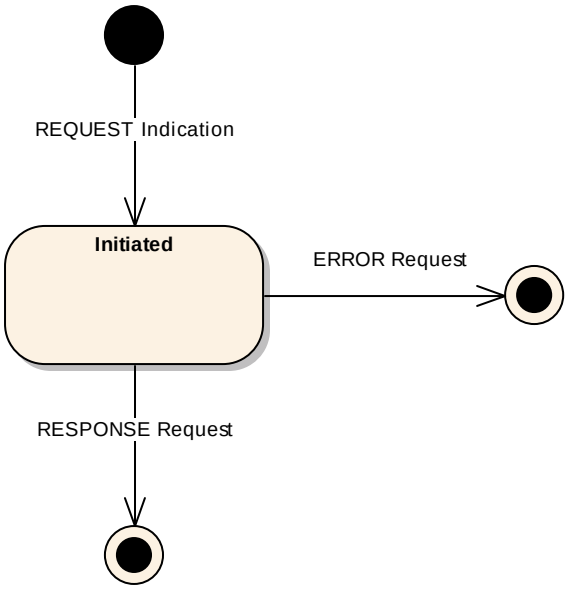

Figure 3-9: REQUEST Provider State Chart

a) The initial transition is triggered by the primitive **REQUEST Indication** and shall lead to the **Initiated State**.

b) There are two possible transitions from the **Initiated State**:

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

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

3.5.3.8 Requests and Indications

3.5.3.8.1 REQUEST

3.5.3.8.1.1 Function

a) The **REQUEST Request** primitive shall be used by the consumer application to initiate a REQUEST Interaction.

b) The **REQUEST Indication** primitive shall be used by the provider MAL to deliver a **REQUEST Message** to a provider and initiate a REQUEST Interaction.

3.5.3.8.1.2 Semantics

REQUEST Request and **REQUEST Indication** shall provide parameters as follows:

(REQUEST Message Header, REQUEST Message Body, QoS properties)

3.5.3.8.1.3 When Generated

a) A **REQUEST Request** may be generated by the consumer application at any time.

b) 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.

3.5.3.8.1.4 Effect on Reception

a) 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.

b) The consumer MAL shall enter the **Initiated State** at this point.

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

d) The provider shall then start processing the operation at this point.

3.5.3.8.1.5 Message Header

a) For the **REQUEST Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-11.

b) 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
URI From	Consumer URI
Authentication Id	Consumer Authentication Identifier
URI To	Provider URI
Session	Session of message
Interaction Type	REQUEST
Interaction Stage	1
Transaction Id	Provided by consumer MAL

3.5.3.8.2 RESPONSE

3.5.3.8.2.1 Function

a) The **RESPONSE Request** primitive shall be used by the provider application to transmit a response to a REQUEST Interaction.

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

3.5.3.8.2.2 Semantics

RESPONSE Request and **RESPONSE Indication** shall provide parameters as follows:

(RESPONSE Message Header, RESPONSE Message Body, QoS properties)

3.5.3.8.2.3 When Generated

a) 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.

b) 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.

3.5.3.8.2.4 Effect on Reception

a) 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.

b) The pattern shall end at this point for the provider.

c) On reception of a **RESPONSE Indication** a consumer shall end the interaction pattern with success.

3.5.3.8.2.5 Message Header

a) For the **RESPONSE Message,** the message header fields shall be as defined in 3.4 except for the fields in table 3-12.

b) 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
URI From	Provider URI
Authentication Id	Provider Authentication Identifier
URI To	Consumer URI
Interaction Type	REQUEST
Interaction Stage	2
Transaction Id	Transaction Id from initial message

3.5.3.8.3 ERROR

3.5.3.8.3.1 Function

a) The **ERROR Request** primitive shall be used by the provider application to end a REQUEST Interaction with an error.

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

3.5.3.8.3.2 Semantics

ERROR Request and **ERROR Indication** shall provide parameters as follows:

(ERROR Message Header, StandardError, QoS properties)

3.5.3.8.3.3 When Generated

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

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

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

2) an error raised by the local communication layer.

3.5.3.8.3.4 Effect on Reception

a) 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.

b) The pattern shall end at this point for the provider.

c) On reception of an **ERROR Indication** a consumer shall end the interaction with an error.

3.5.3.8.3.5 Message Header

a) 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.

b) The contents of the StandardError structure shall immediately follow the message header.

3.5.3.9 Example

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

Operation Identifier	testRequest	
Interaction Pattern	REQUEST	
Pattern Sequence	Message	Body Type
IN	REQUEST	TestBody
OUT	RESPONSE	TestResponse

The TestBody structure is defined in 3.5.1.9 and the TestResponse structure is defined below:

Structure Name	TestResponse	
Extends	Composite	
Short form	Example Only	
Field	Type	Comment
RspnItem	Boolean	Example Boolean item
RspnField	Float	Example Float item

This corresponds to the following message being transmitted:

Message Header																*Test Body*	
URI From	*Auth Id*	*URI To*	*Timestamp*	*QoS*	*Priority*	*Domain*	*Zone*	*Session*	*Interaction*	*Stage*	*Trans Id*	*Area*	*Service*	*Operation*	*Version*	*Is Error*	*First Item*	*Second Item*
Consumer	Op A	Provider		BEST	1	A.B.C	GROUND	LIVE	REQUEST	1	1235	Example	Example	testRequest	1	FALSE	Hello	1234

And corresponds to the following response being returned:

Message Header																*Test Response*	
URI From	*Auth Id*	*URI To*	*Timestamp*	*QoS*	*Priority*	*Domain*	*Zone*	*Session*	*Interaction*	*Stage*	*Trans Id*	*Area*	*Service*	*Operation*	*Version*	*Is Error*	*Rspn Item*	*Rspn Field*
Provider	SC X	Consumer		BEST	1	A.B.C	GROUND	LIVE	REQUEST	2	1235	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.

3.5.4 INVOKE INTERACTION PATTERN

3.5.4.1 Overview

The INVOKE pattern extends the REQUEST pattern to add a mandatory acknowledgement of the initial message (figure 3-10). This allows the operation to confirm the receipt of the request before proceeding to process the request.

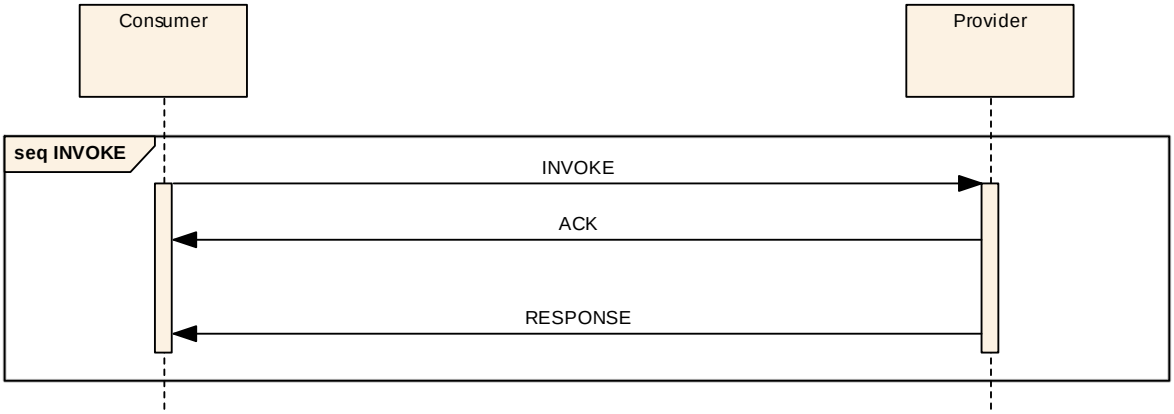

Figure 3-10: INVOKE Interaction Pattern Message Sequence

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.

3.5.4.2 Description

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

3.5.4.3 Usage

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

3.5.4.4 Error Handling

The INVOKE pattern can report failure in two distinct ways:

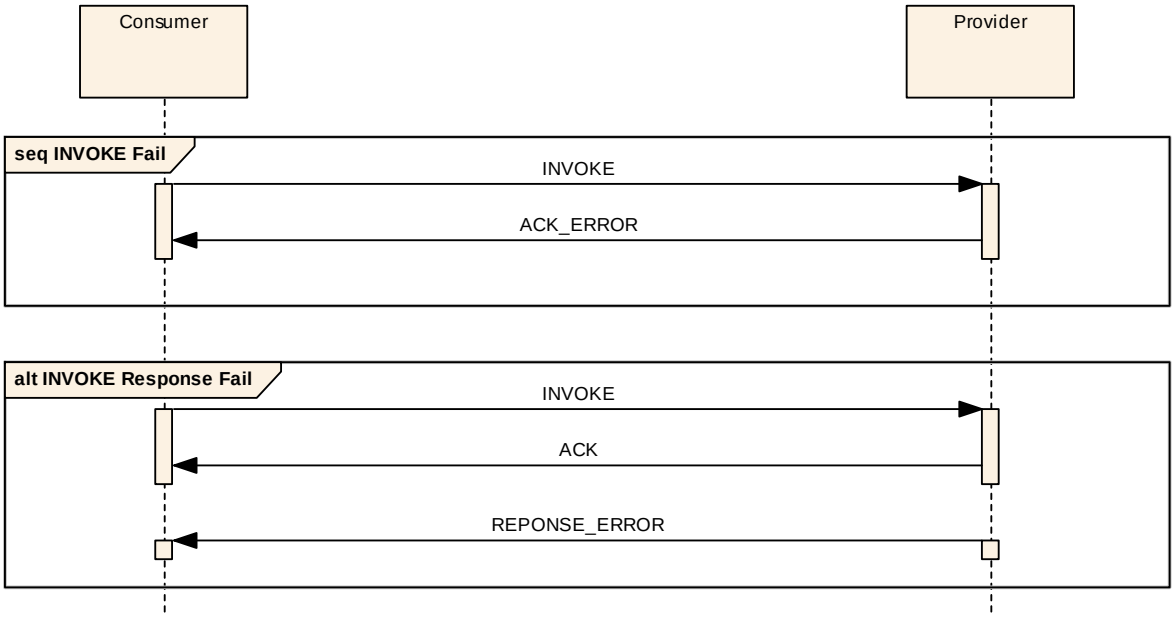

Figure 3-11: INVOKE Interaction Pattern Error Sequence

a) The provider may return an error message (see 4.4.6) in replacement of the ACK message, shown in the first sequence in figure 3-11.

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 3-11.

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.

3.5.4.5 Operation Template

The INVOKE template extends the REQUEST pattern template by requiring a structure for the acknowledgment message. INVOKE operations take a single message body argument and return an acknowledgment structure and a single data response:

Table 3-13: INVOKE Operation Template

Operation Identifier	<<Operation name>>	
Interaction Pattern	INVOKE	
Pattern Sequence	Message	Body Type
IN	INVOKE	<<Request type>>
OUT	ACK	<<Ack type>>
OUT	RESPONSE	<<Response type>>

3.5.4.6 Primitives

Table 3-14: INVOKE Primitive List

Primitive
INVOKE
ACK
RESPONSE
ACK_ERROR
RESPONSE_ERROR

3.5.4.7 State Charts

3.5.4.7.1 Consumer Side

Figure 3-12 shows the consumer side state chart for the INVOKE pattern:

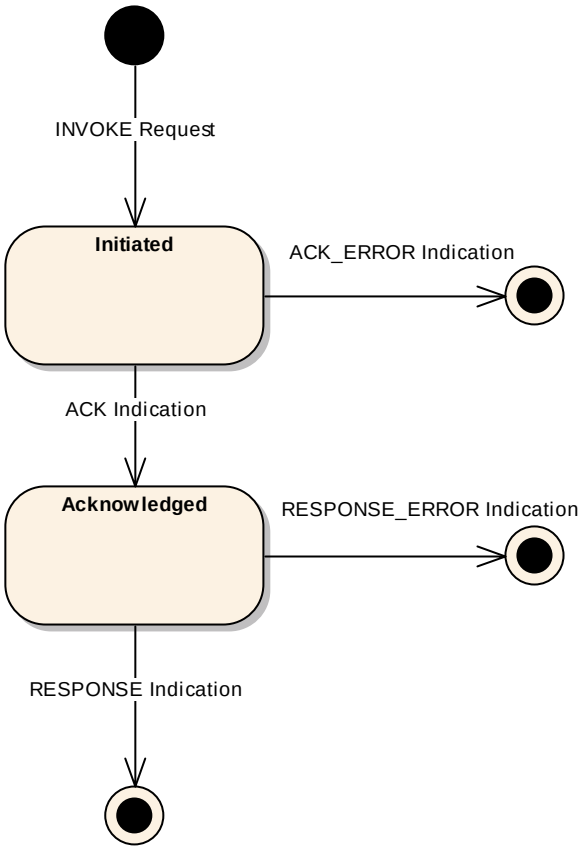

Figure 3-12: INVOKE Consumer State Chart

a) The initial transition is triggered by the primitive **INVOKE Request** and shall lead to the **Initiated State**.

b) There are two possible transitions from the **Initiated State**:

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

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

c) There are two possible transitions from the **Acknowledged State**:

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

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

3.5.4.7.2 Provider Side

Figure 3-13 shows the provider side state chart for the INVOKE pattern:

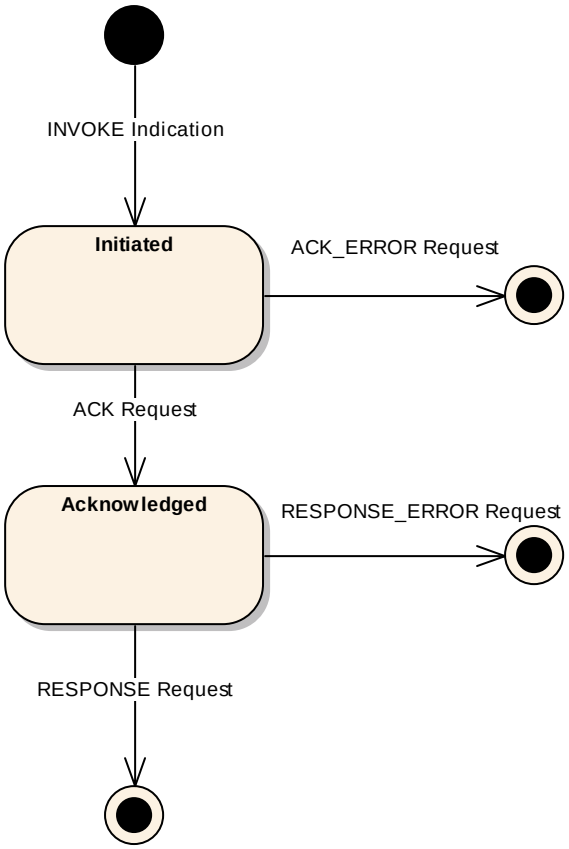

Figure 3-13: INVOKE Provider State Chart

a) The initial transition is triggered by the primitive **INVOKE Indication** and shall lead to the **Initiated State**.

b) There are two possible transitions from the **Initiated State**:

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

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

c) There are two possible transitions from the **Acknowledged State**:

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

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

3.5.4.8 Requests and Indications

3.5.4.8.1 INVOKE

3.5.4.8.1.1 Function

a) The **INVOKE Request** primitive shall be used by the consumer application to initiate an INVOKE Interaction.

b) The **INVOKE Indication** primitive shall be used by the provider MAL to deliver an **INVOKE Message** to a provider and initiate an INVOKE Interaction.

3.5.4.8.1.2 Semantics

INVOKE Request and **INVOKE Indication** shall provide parameters as follows:

(INVOKE Message Header, INVOKE Message Body, QoS properties)

3.5.4.8.1.3 When Generated

a) An **INVOKE Request** may be generated by the consumer application at any time.

b) 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.

3.5.4.8.1.4 Effect on Reception

a) 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.

b) The consumer MAL shall enter the **Initiated State**.

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

d) The provider shall then start processing the operation at this point.

3.5.4.8.1.5 Message Header

a) For the **INVOKE Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-15.

b) 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
URI From	Consumer URI
Authentication Id	Consumer Authentication Identifier
URI To	Provider URI
Interaction Type	INVOKE
Interaction Stage	1
Transaction Id	Provided by consumer MAL

3.5.4.8.2 ACK

3.5.4.8.2.1 Function

a) The **ACK Request** primitive shall be used by the provider application to acknowledge an INVOKE Interaction.

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

3.5.4.8.2.2 Semantics

ACK Request and **ACK Indication** shall provide parameters as follows:

(ACK Message Header, **A**CK Message Body, QoS properties)

3.5.4.8.2.3 When Generated

a) An **ACK Request** shall be used by the provider application with the provider MAL in the **Initiated State** to acknowledge the INVOKE Interaction.

b) 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.

3.5.4.8.2.4 Effect on Reception

a) 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.

b) A provider MAL in the **Initiated State** shall enter the **Acknowledged State**.

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

3.5.4.8.2.5 Message Header

a) For the **ACK Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-16.

b) 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
URI From	Provider URI
Authentication Id	Provider Authentication Identifier
URI To	Consumer URI
Interaction Type	INVOKE
Interaction Stage	2
Transaction Id	Transaction Id from initial message

3.5.4.8.3 ACK_ERROR

3.5.4.8.3.1 Function

a) The **ACK_ERROR Request** primitive shall be used by the provider application to end an INVOKE Interaction with an error.

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

3.5.4.8.3.2 Semantics

ACK_ERROR Request and **ACK_ERROR Indication** shall provide parameters as follows:

(ACK_ERROR Message Header, StandardError, QoS properties)

3.5.4.8.3.3 When Generated

a) **ACK_ERROR Request** shall be used by the provider application with the provider MAL in the **Initiated State** to transmit an error.

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

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

   2) an error raised by the local communication layer.

3.5.4.8.3.4 Effect on Reception

a) 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.

b) The pattern shall end at this point for the provider.

c) On reception of an **ACK_ERROR Indication** a consumer shall end the interaction with an error.

3.5.4.8.3.5 Message Header

a) 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.

b) The contents of the StandardError structure shall immediately follow the message header.

3.5.4.8.4 RESPONSE

3.5.4.8.4.1 Function

a) The **RESPONSE Request** primitive shall be used by the provider application to transmit a response to an INVOKE Interaction.

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

3.5.4.8.4.2 Semantics

RESPONSE Request and **RESPONSE Indication** shall provide parameters as follows:

(RESPONSE Message Header, RESPONSE Message Body, QoS properties)

3.5.4.8.4.3 When Generated

a) 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.

b) 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.

3.5.4.8.4.4 Effect on Reception

a) 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.

b) The pattern shall end at this point for the provider.

c) On reception of a **RESPONSE Indication** a consumer shall end the INVOKE Interaction with success.

3.5.4.8.4.5 Message Header

a) For the **RESPONSE Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-17.

b) 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
URI From	Provider URI
Authentication Id	Provider Authentication Identifier
URI To	Consumer URI
Interaction Type	INVOKE
Interaction Stage	3
Transaction Id	Transaction Id from initial message

3.5.4.8.5 RESPONSE_ERROR

3.5.4.8.5.1 Function

a) The **RESPONSE_ERROR Request** primitive shall be used by the provider application to end an INVOKE Interaction with an error.

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

3.5.4.8.5.2 Semantics

RESPONSE_ERROR Request and **RESPONSE_ERROR Indication** shall provide parameters as follows:

(RESPONSE_ERROR Message Header, StandardError, QoS properties)

3.5.4.8.5.3 When Generated

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

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

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

2) an error raised by the local communication layer.

3.5.4.8.5.4 Effect on Reception

a) 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.

b) The pattern shall end at this point for the provider.

c) On reception of a **RESPONSE_ERROR Indication** a consumer shall end the interaction with an error.

3.5.4.8.5.5 Message Header

a) 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.

b) The contents of the StandardError structure shall immediately follow the message header.

3.5.4.9 Example

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	Body Type
IN	INVOKE	TestBody
OUT	ACK	TestAck
OUT	RESPONSE	TestResponse

The TestBody structure is defined in 3.5.1.9 and the TestResponse structure is defined in 3.5.3.9. The TestAck structure is defined below:

Structure Name	TestAck	
Extends	Composite	
Short form	Example Only	
Field	Type	Comment
AckId	Identifier	Example Identifier item

This corresponds to the following message being transmitted:

Message Header																*Test Body*	
URI From	*Auth Id*	*URI To*	*Timestamp*	*QoS*	*Priority*	*Domain*	*Zone*	*Session*	*Interaction*	*Stage*	*Trans Id*	*Area*	*Service*	*Operation*	*Version*	*Is Error*	*First Item*	*Second Item*
Consumer	Op A	Provider		BEST	1	A.B.C	GROUND	LIVE	INVOKE	1	1236	Example	Example	testInvoke	1	FALSE	Hello	1234

And corresponds to the following acknowledgement being returned:

Message Header																*Test Ack*
URI From	*Auth Id*	*URI To*	*Timestamp*	*QoS*	*Priority*	*Domain*	*Zone*	*Session*	*Interaction*	*Stage*	*Trans Id*	*Area*	*Service*	*Operation*	*Version*	*Is Error*	*Ack Id*
Provider	SC X	Consumer		BEST	1	A.B.C	GROUND	LIVE	INVOKE	2	1236	Example	Example	testInvoke	1	FALSE	1234

And finally corresponds to the following response being returned:

Message Header																*Test Response*	
URI From	*Auth Id*	*URI To*	*Timestamp*	*QoS*	*Priority*	*Domain*	*Zone*	*Session*	*Interaction*	*Stage*	*Trans Id*	*Area*	*Service*	*Operation*	*Version*	*Is Error*	*Rspn Item*	*Rspn Field*
Provider	SC X	Consumer		BEST	1	A.B.C	GROUND	LIVE	INVOKE	3	1236	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.

3.5.5 PROGRESS INTERACTION PATTERN

3.5.5.1 Overview

The PROGRESS pattern extends the INVOKE pattern to add a set of mandatory progress updates of the request message (figure 3-14). This allows the operation to confirm the receipt of the request before proceeding to process the request and also to show progress of the operation.

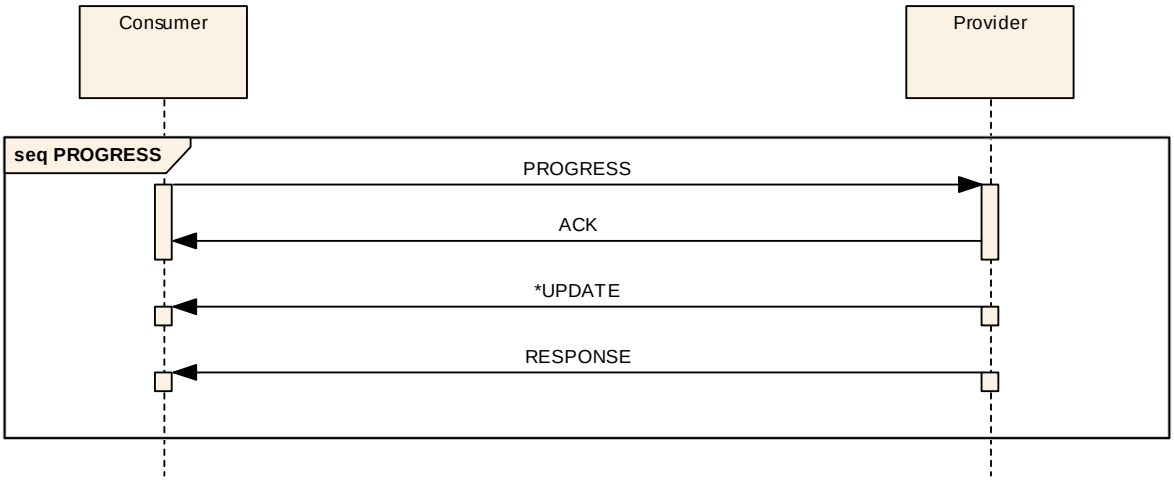

Figure 3-14: PROGRESS Interaction Pattern Message Sequence

3.5.5.2 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 is defined by the service and not by the pattern.

3.5.5.3 Usage

The PROGRESS pattern is expected to 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.

3.5.5.4 Error Handling

The PROGRESS pattern can report failure in three distinct ways:

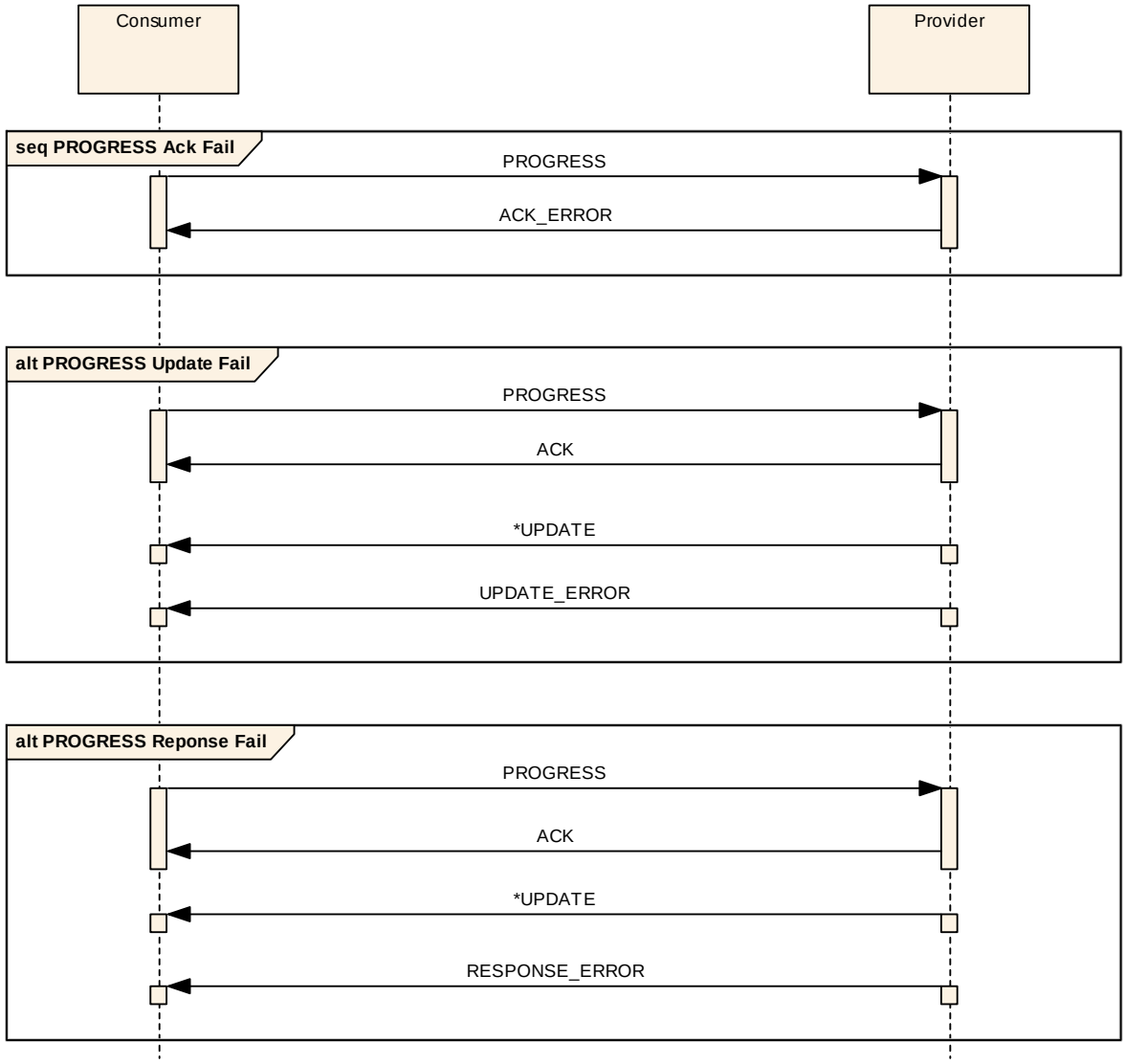

Figure 3-15: PROGRESS Interaction Pattern Error Sequence

a) The acknowledgement may be replaced with an error message (see 4.4.6) shown in the first sequence in figure 3-15.

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 3-15 and replaces any of the progress updates.

c) An error may be generated after the updates and replace the final response as shown in the third sequence in figure 3-15.

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

3.5.5.5 Operation Template

The PROGRESS template is similar to the INVOKE pattern template but adds a progress update message:

Table 3-18: PROGRESS Operation Template

Operation Identifier	<<Operation name>>	
Interaction Pattern	PROGRESS	
Pattern Sequence	Message	Body Type
IN	PROGRESS	<<Request type>>
OUT	ACK	<<Ack type>>
OUT	UPDATE	<<Update type>>
OUT	RESPONSE	<<Response type>>

3.5.5.6 Primitives

Table 3-19: PROGRESS Primitive List

Primitive
PROGRESS
ACK
ACK_ERROR
UPDATE
UPDATE_ERROR
RESPONSE
RESPONSE_ERROR

3.5.5.7 State Charts

3.5.5.7.1 Consumer Side

Figure 3-16 shows the consumer side state chart for the PROGRESS pattern:

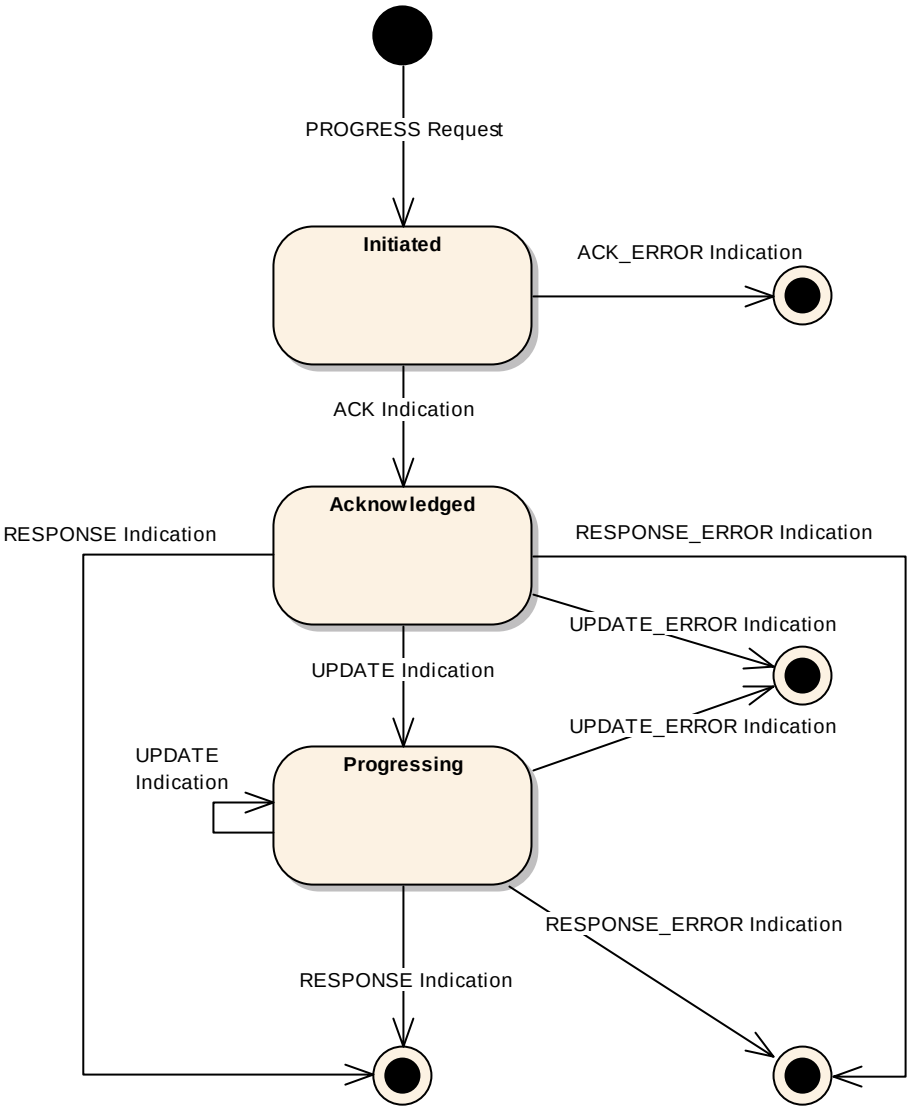

Figure 3-16: PROGRESS Consumer State Chart

a) The initial transition is triggered by the primitive **PROGRESS Request** and shall lead to the **Initiated State**.

b) There are two possible transitions from the **Initiated State**:

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

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

c) There are four possible transitions from the **Acknowledged State**:

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

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

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

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

d) There are four possible transitions from the **Progressing State**:

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

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

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

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

3.5.5.7.2 Provider Side

Figure 3-17 shows the provider side state chart for the PROGRESS pattern:

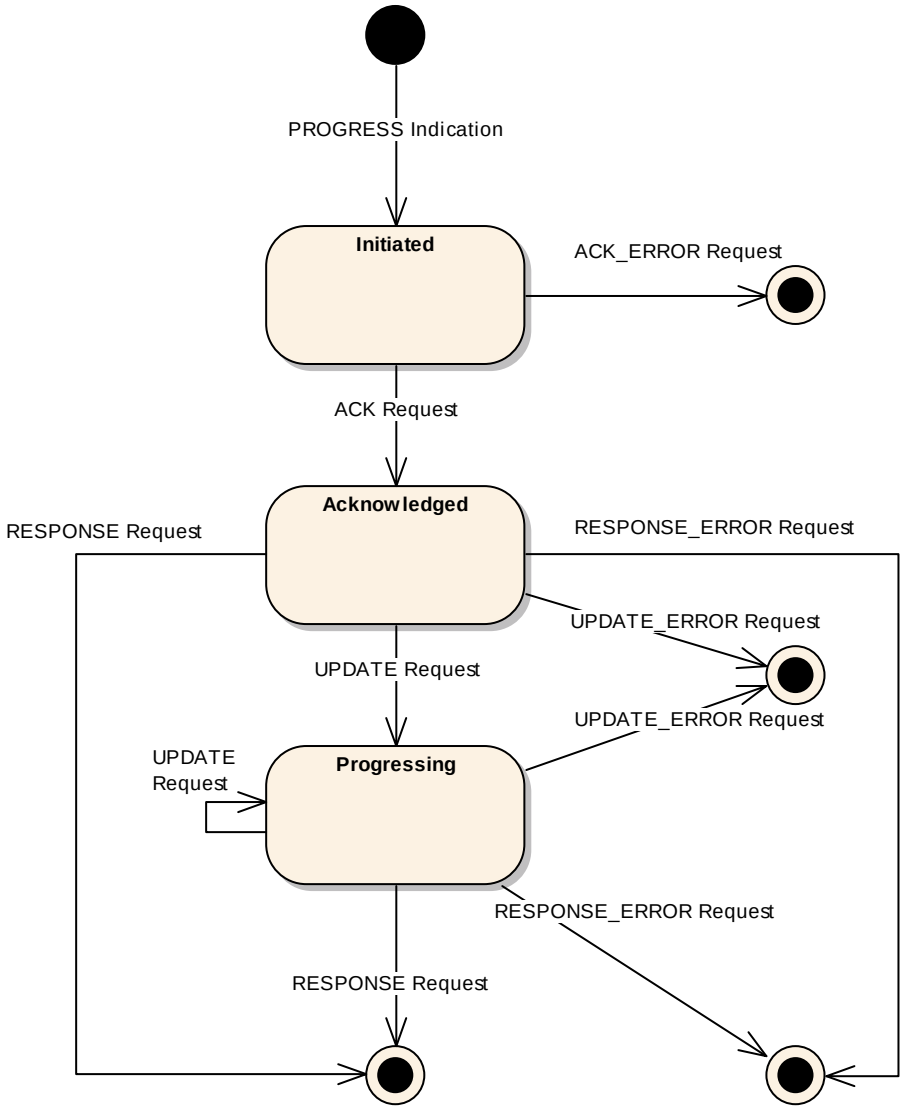

Figure 3-17: PROGRESS Provider State Chart

a) The initial transition is triggered by the primitive **PROGRESS Indication** and shall lead to the **Initiated State**.

b) There are two possible transitions from the **Initiated State**:

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

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

c) There are four possible transitions from the **Acknowledged State**:

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

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

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

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

d) There are four possible transitions from the **Progressing State**:

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

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

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

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

3.5.5.8 Requests and Indications

3.5.5.8.1 PROGRESS

3.5.5.8.1.1 Function

a) The **PROGRESS Request** primitive shall be used by the consumer application to initiate a PROGRESS Interaction.

b) The **PROGRESS Indication** primitive shall be used by the provider MAL to deliver a **PROGRESS Message** to a provider and initiate a PROGRESS Interaction.

3.5.5.8.1.2 Semantics

PROGRESS Request and **PROGRESS Indication** shall provide parameters as follows:

(PROGRESS Message Header, PROGRESS Message Body, QoS properties)

3.5.5.8.1.3 When Generated

a) A **PROGRESS Request** may be generated by the consumer application at any time.

b) 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.

3.5.5.8.1.4 Effect on Reception

a) 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.

b) The consumer MAL shall enter the **Initiated State**.

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

d) The provider shall then start processing the operation at this point.

3.5.5.8.1.5 Message Header

a) For the **PROGRESS Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-20.

b) 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
URI From	Consumer URI
Authentication Id	Consumer Authentication Identifier
URI To	Provider URI
Interaction Type	PROGRESS
Interaction Stage	1
Transaction Id	Provided by consumer MAL

3.5.5.8.2 ACK

3.5.5.8.2.1 Function

a) The **ACK Request** primitive shall be used by the provider application to acknowledge a PROGRESS Interaction.

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

3.5.5.8.2.2 Semantics

ACK Request and **ACK Indication** shall provide parameters as follows:

(ACK Message Header, **A**CK Message Body, QoS properties)

3.5.5.8.2.3 When Generated

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

b) 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.

3.5.5.8.2.4 Effect on Reception

a) 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.

b) A provider MAL in the **Initiated State** shall enter the **Acknowledged State**.

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

3.5.5.8.2.5 Message Header

a) For the **ACK Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-21.

b) 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
URI From	Provider URI
Authentication Id	Provider Authentication Identifier
URI To	Consumer URI
Interaction Type	PROGRESS
Interaction Stage	2
Transaction Id	Transaction Id from initial message

3.5.5.8.3 ACK_ERROR

3.5.5.8.3.1 Function

a) The **ACK_ERROR Request** primitive shall be used by the provider application to end a PROGRESS Interaction with an error.

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

3.5.5.8.3.2 Semantics

ACK_ERROR Request and **ACK_ERROR Indication** shall provide parameters as follows:

(ACK_ERROR Message Header, StandardError, QoS properties)

3.5.5.8.3.3 When Generated

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

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

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

2) an error raised by the local communication layer.

3.5.5.8.3.4 Effect on Reception

a) 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.

b) The pattern shall end at this point for the provider.

c) On reception of an **ACK_ERROR Indication** a consumer shall end the interaction with an error.

3.5.5.8.3.5 Message Header

a) 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.

b) The contents of the StandardError structure shall immediately follow the message header.

3.5.5.8.4 UPDATE

3.5.5.8.4.1 Function

a) The **UPDATE Request** primitive shall be used by the provider application to transmit an update during a PROGRESS Interaction.

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

3.5.5.8.4.2 Semantics

UPDATE Request and **UPDATE Indication** shall provide parameters as follows:

(UPDATE Message Header, UPDATE Message Body, QoS properties)

3.5.5.8.4.3 When Generated

a) 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.

b) 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.

3.5.5.8.4.4 Effect on Reception

a) 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.

b) A provider MAL in the **Acknowledged State** shall enter the **Progressing State**.

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

3.5.5.8.4.5 Message Header

a) For the **UPDATE Message**, of which there may be several, the message header fields shall be as defined in 3.4 except for the fields in table 3-21.

b) 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
URI From	Provider URI
Authentication Id	Provider Authentication Identifier
URI To	Consumer URI
Interaction Type	PROGRESS
Interaction Stage	3
Transaction Id	Transaction Id from initial message

3.5.5.8.5 UPDATE_ERROR

3.5.5.8.5.1 Function

a) The **UPDATE_ERROR Request** primitive shall be used by the provider application to end a PROGRESS Interaction with an error.

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

3.5.5.8.5.2 Semantics

UPDATE_ERROR Request and **UPDATE_ERROR Indication** shall provide parameters as follows:

(UPDATE_ERROR Message Header, StandardError, QoS properties)

3.5.5.8.5.3 When Generated

a) 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.

b) 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:

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

2) an error raised by the local communication layer.

3.5.5.8.5.4 Effect on Reception

a) 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.

b) The pattern shall end at this point for the provider.

c) On reception of an **UPDATE_ERROR Indication** a consumer shall end the interaction with an error.

3.5.5.8.5.5 Message Header

a) 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.

b) The contents of the StandardError structure shall immediately follow the message header.

3.5.5.8.6 RESPONSE

3.5.5.8.6.1 Function

a) The **RESPONSE Request** primitive shall be used by the provider application to transmit a response to a PROGRESS Interaction.

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

3.5.5.8.6.2 Semantics

RESPONSE Request and **RESPONSE Indication** shall provide parameters as follows:

(RESPONSE Message Header, RESPONSE Message Body, QoS properties)

3.5.5.8.6.3 When Generated

a) 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.

b) 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.

3.5.5.8.6.4 Effect on Reception

a) 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.

b) The pattern shall end at this point for the provider.

c) On reception of a **RESPONSE Indication** a consumer shall end the PROGRESS Interaction with success.

3.5.5.8.6.5 Message Header

a) For the **RESPONSE Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-23.

b) 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
URI From	Provider URI
Authentication Id	Provider Authentication Identifier
URI To	Consumer URI
Interaction Type	PROGRESS
Interaction Stage	4
Transaction Id	Transaction Id from initial message

3.5.5.8.7 RESPONSE_ERROR

3.5.5.8.7.1 Function

a) The **RESPONSE_ERROR Request** primitive shall be used by the provider application to end a PROGRESS Interaction with an error.

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

3.5.5.8.7.2 Semantics

RESPONSE_ERROR Request and **RESPONSE_ERROR Indication** shall provide parameters as follows:

(RESPONSE_ERROR Message Header, StandardError, QoS properties)

3.5.5.8.7.3 When Generated

a) 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.

b) 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:

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

   2) an error raised by the local communication layer.

3.5.5.8.7.4 Effect on Reception

a) 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.

b) The pattern shall end at this point for the provider.

c) On reception of a **RESPONSE_ERROR Indication** a consumer shall end the interaction with an error.

3.5.5.8.7.5 Message Header

a) 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.

b) The contents of the StandardError structure shall immediately follow the message header.

3.5.5.9 Example

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	Body Type
IN	PROGRESS	TestBody
OUT	ACK	TestAck
OUT	UPDATE	TestProgress
OUT	RESPONSE	TestResponse

The TestBody structure is defined in 3.5.1.9, TestAck structure is defined in 3.5.4.9, and the TestResponse structure is defined in 3.5.3.9. The TestProgress structure is defined below:

Structure Name	TestProgress	
Extends	Composite	
Short form	Example Only	
Field	Type	Comment
RspnStage	Integer	Example Integer item

This corresponds to the following message being transmitted:

Message Header																	*Test Body*	
URI From	*Auth Id*	*URI To*	*Timestamp*	*Domain*	*QoS*	*Priority*	*Zone*	*Session*	*Interaction*	*Stage*	*Trans Id*	*Area*	*Service*	*Operation*	*Version*	*Is Error*	*First Item*	*Second Item*
Consumer	Op A	Provider		A.B.C	BEST	1	GROUND	LIVE	PROGRESS	1	1237	Example	Example	testProg	1	FALSE	Hello	1234

And corresponds to the following acknowledgement being returned:

Message Header																Test Ack
URI From	Auth Id	URI To	Timestamp	Domain	QoS	Priority	Zone	Session	Interaction	Stage	Trans Id	Area	Service	Operation	Version	Is Error	Ack Id
Provider	SC X	Consumer		A.B.C	BEST	1	GROUND	LIVE	PROGRESS	2	1237	Example	Example	testProg	1	FALSE	1234

Progress is reported using the following message; in this example two stages are reported:

Message Header																Test Progress
URI From	Auth Id	URI To	Timestamp	Domain	QoS	Priority	Zone	Session	Interaction	Stage	Trans Id	Area	Service	Operation	Version	Is Error	Rspn Stage
Provider	SC X	Consumer		A.B.C	BEST	1	GROUND	LIVE	PROGRESS	3	1237	Example	Example	testProg	1	FALSE	1

Message Header																Test Progress
URI From	Auth Id	URI To	Timestamp	Domain	QoS	Priority	Zone	Session	Interaction	Stage	Trans Id	Area	Service	Operation	Version	Is Error	Rspn Stage
Provider	SC X	Consumer		A.B.C	BEST	1	GROUND	LIVE	PROGRESS	3	1237	Example	Example	testProg	1	FALSE	2

And finally corresponds to the following response being returned:

Message Header																Test Response	
URI From	Auth Id	URI To	Timestamp	Domain	QoS	Priority	Zone	Session	Interaction	Stage	Trans Id	Area	Service	Operation	Version	Is Error	Rspn Item	Rspn Field
Provider	SC X	Consumer		A.B.C	BEST	1	GROUND	LIVE	PROGRESS	4	1237	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.

3.5.6 PUBLISH-SUBSCRIBE INTERACTION PATTERN

3.5.6.1 Overview

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 consumers register interest by sending a subscription list; providers also register the list of entities they provide; providers publish updates; updates are then sent to registered consumers; and when no further updates are required the consumer cancels its subscription by deregistering.

The PUBLISH-SUBSCRIBE pattern can be deployed in two distinct ways; the first is where an intermediary shared message broker resides between the consumers and the providers (figure 3-18). In this deployment 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:

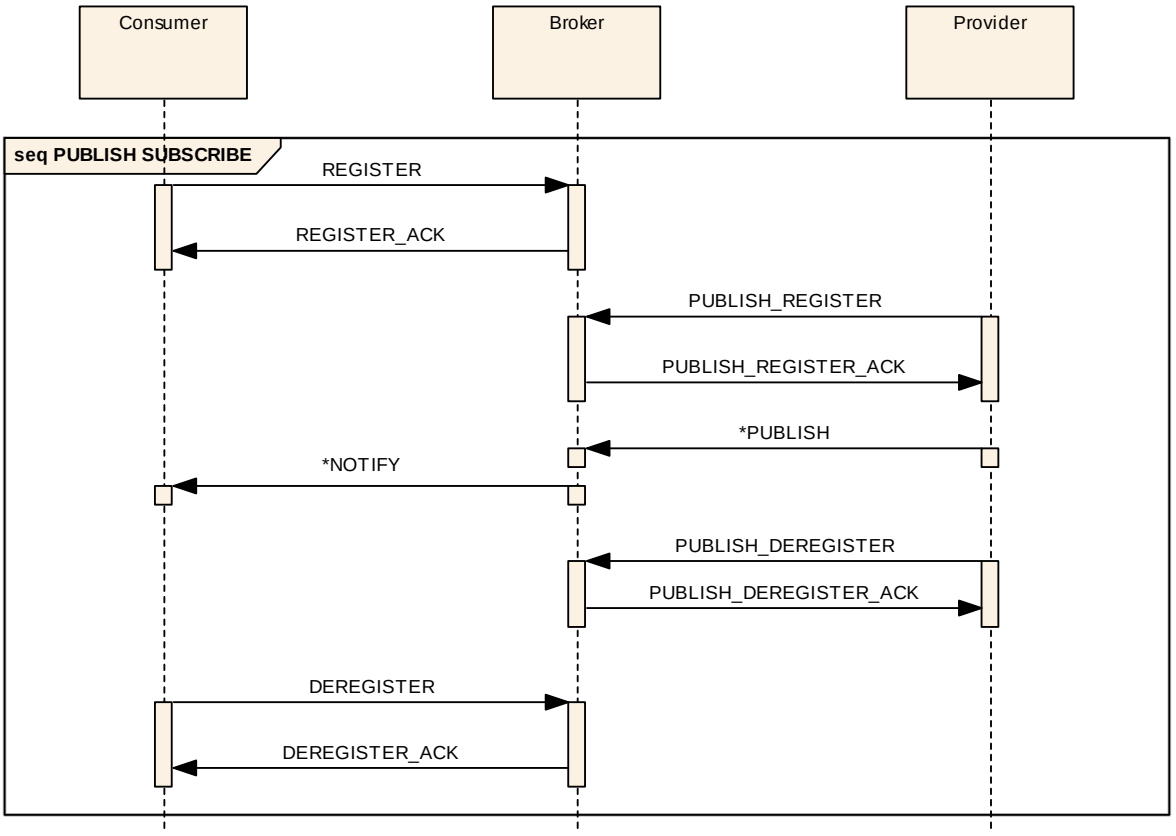

Figure 3-18: PUBLISH-SUBSCRIBE Interaction Pattern Message Sequence

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.

The second deployment of the PUBLISH-SUBSCRIBE pattern is when there is a single update provider and the message broker is private to it. This is an example of an observe type relationship where 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 3-19):

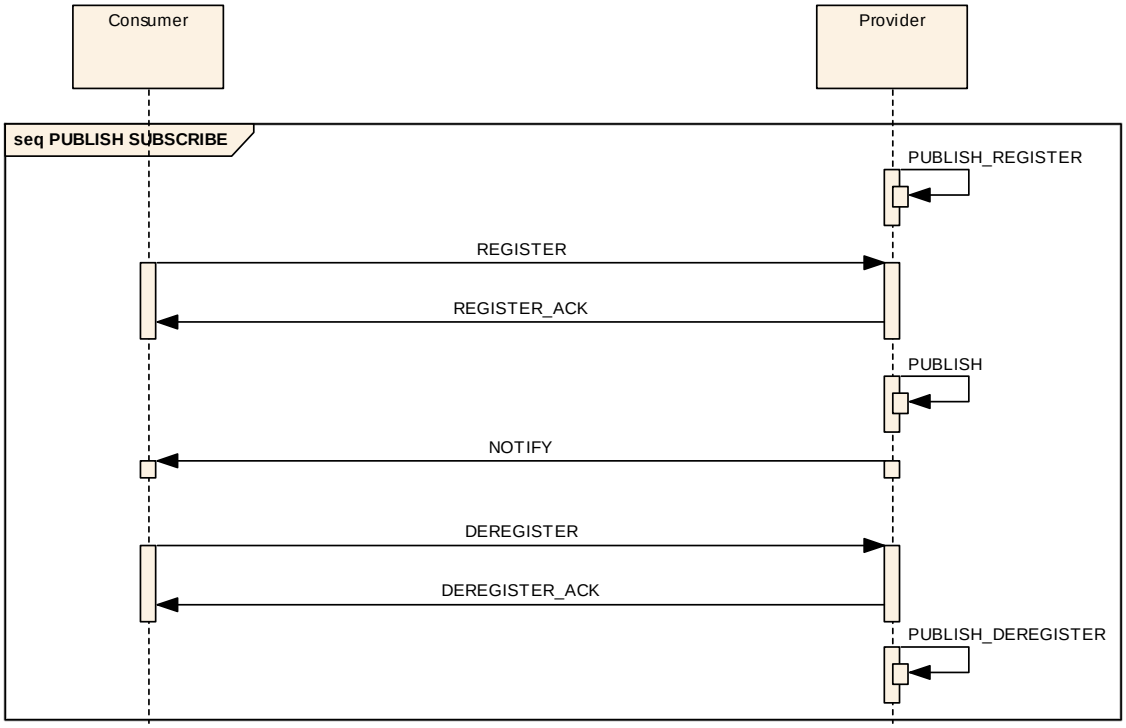

Figure 3-19: PUBLISH-SUBSCRIBE Pattern Alternative Message Sequence

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

3.5.6.2 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.

3.5.6.3 Subscriptions

A consumer informs the broker of the updates to send it by the use of subscriptions. Subscriptions are sent during the registration.

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

b) 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.

c) 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.

d) The URI 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 consumer URIs.

e) The consumer deregister message takes a list of identifiers of the subscription lists to cancel and shall completely remove the subscriptions from the list of active subscriptions.

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

For example, if a consumer initially registers for updates on the set (A, B, C) but at a later date requires also ‘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.

Another way of describing this is the register message defines an update ‘report’ and the deregister message clears that ‘report’ definition.

NOTE – 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.

3.5.6.4 Entity Identification

The identity of an entity is composed of two parts: specific fields from the message header and an Entity Key:

a) The domain, session type and name, area, service, and operation identifiers of the message header shall form the first part of the Entity identification.

b) The network identifier shall be ignored and does not form part of the Entity identification.

c) The Entity Key is a composite structure that contains four sub-keys and shall form the second part of the Entity identification.

d) The combination of the message header fields and the Entity Key in a message shall form the complete Entity identification.

3.5.6.5 Subscription Matching

This subsection details the rules for subscription matching in the PUBLISH-SUBSCRIBE pattern. The subscription is matched on two aspects: the Entity Key, the optional subDomain field of the Entity Request, and the message header fields.

The following paragraphs detail the rules for Entity Key matching:

a) A sub-key specified in the EntityKey structure shall take one of three types of value: an actual value, a NULL value, and the special wildcard identifier of ‘*’.

b) If a sub-key contains a specific value it shall only match a sub-key that contains the same value. This includes an empty ‘’ value. The matches are case sensitive.

c) If a sub-key contains a NULL value it shall only match a sub-key that contains NULL.

d) If a sub-key contains a wildcard identifier it shall match a sub-key that contains any value including NULL.

For example, suppose a provider is publishing a set of updates with the following keys (the dot notation is used to separate the sub-keys; all four sub-keys must be provided):

1) A.[null].[null].[null]

2) A.B.[null].[null]

3) A.B.C.[null]

4) A.B.C.D

5) B.[null].[null].[null]

6) Q.B.C.[null]

An EntityRequest using the key of ‘A.[null].[null].[null]’ would only match to the first update.

An EntityRequest using the key of ‘A.*.[null].[null]’ would only match to the first and second updates.

An EntityRequest using the key of ‘A.*.*.*’ would match to all but the last two updates.

An EntityRequest using the key of ‘A.B.[null].[null]’ would only match the second update.

An EntityRequest using the key of ‘A.B.*.[null]’ would only match to updates 2 and 3.

An EntityRequest using the key of ‘*.B.*.[null]’ would only match to updates 2, 3 and 6.

An EntityRequest using the key of ‘B.*.*.*’ would only match to update 5.

NOTE – The meaning of the entity key value is context-specific and must be detailed in the relevant service specification.

The following parapgraphs detail the rules for message header field matching of the subscription and update message:

e) The domain of the update message shall match the domain of the subscription message.

f) If the subscription EntityRequest included a subDomain field, then this shall be appended to the domain of the subscription message to make the complete domain for that request.

g) The final Identifier of the subDomain may be the wildcard character ‘*’.

h) The session types and names must match.

i) The network identifiers are ignored.

j) The area identifiers must match unless the subscription specified True in the allAreas field of the EntityRequest, in which case they shall be ignored.

k) The service identifiers must match unless the subscription specified True in the allServices field of the EntityRequest, in which case they shall be ignored.

l) The operation identifiers must match unless the subscription specified True in the allOperations field of the EntityRequest, in which case they shall be ignored.

For example, suppose a provider is publishing a set of updates with the following domains (the dot notation is used to separate the sub-domains):

1) spacecraftA

2) spacecraftA.aocs

3) spacecraftA.aocs.thrustA

4) spacecraftA.payload

5) spacecraftA.payload.cameraA.tempB

6) spacecraftB

7) agency.spacecraftA

Therefore a subscription message with the domain of ‘spacecraftA’ and a NULL subDomain field would only match the first update.

A subscription message with the domain of ‘spacecraftA’ and a subDomain field set to ‘aocs’ would only match the second update.

A subscription message with the domain of ‘spacecraftA’ and a subDomain field set to ‘payload.*’ would match the fourth and fifth updates.

A subscription message with the domain of ‘spacecraftA’ and a subDomain field set to ‘*’ would match updates 1 to 5.

3.5.6.6 Usage

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

3.5.6.7 Error Handling

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

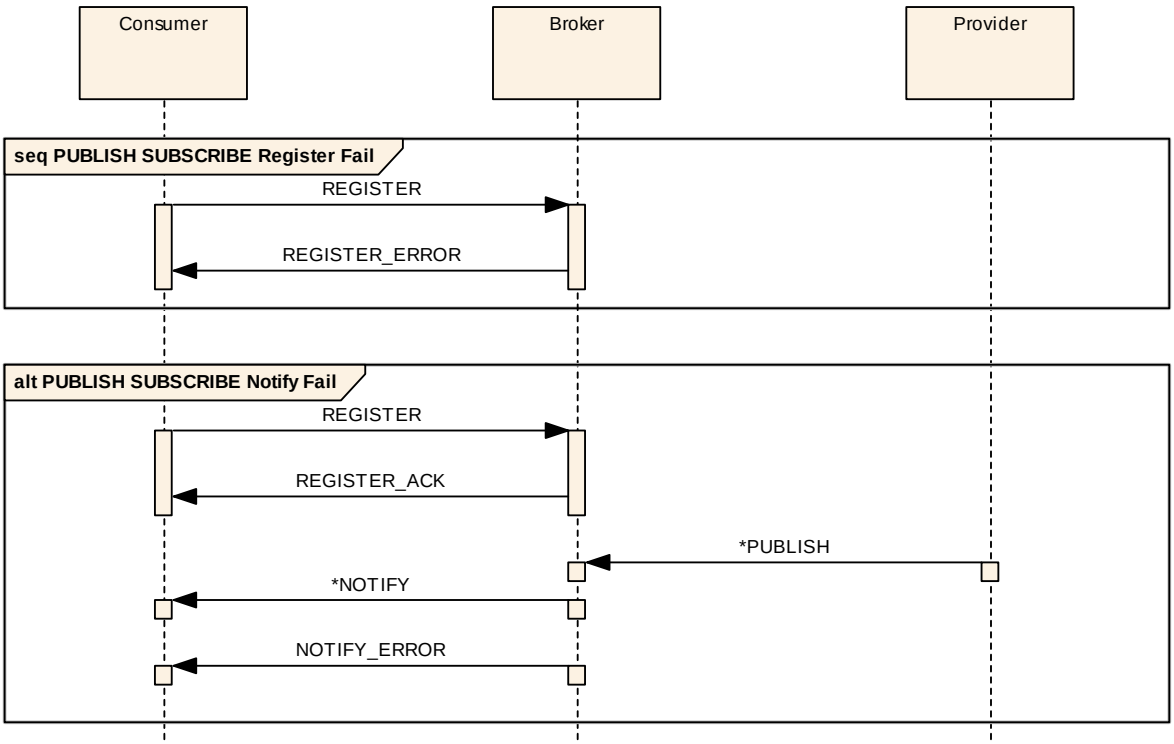

Figure 3-20: PUBLISH-SUBSCRIBE Interaction Pattern Consumer Error Sequence

a) The register acknowledgement may be replaced with an error message (see 4.4.6) shown in the first sequence in figure 3-20.

b) A notify may be replaced by an error by the message broker shown in the second sequence in figure 3-20.

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

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

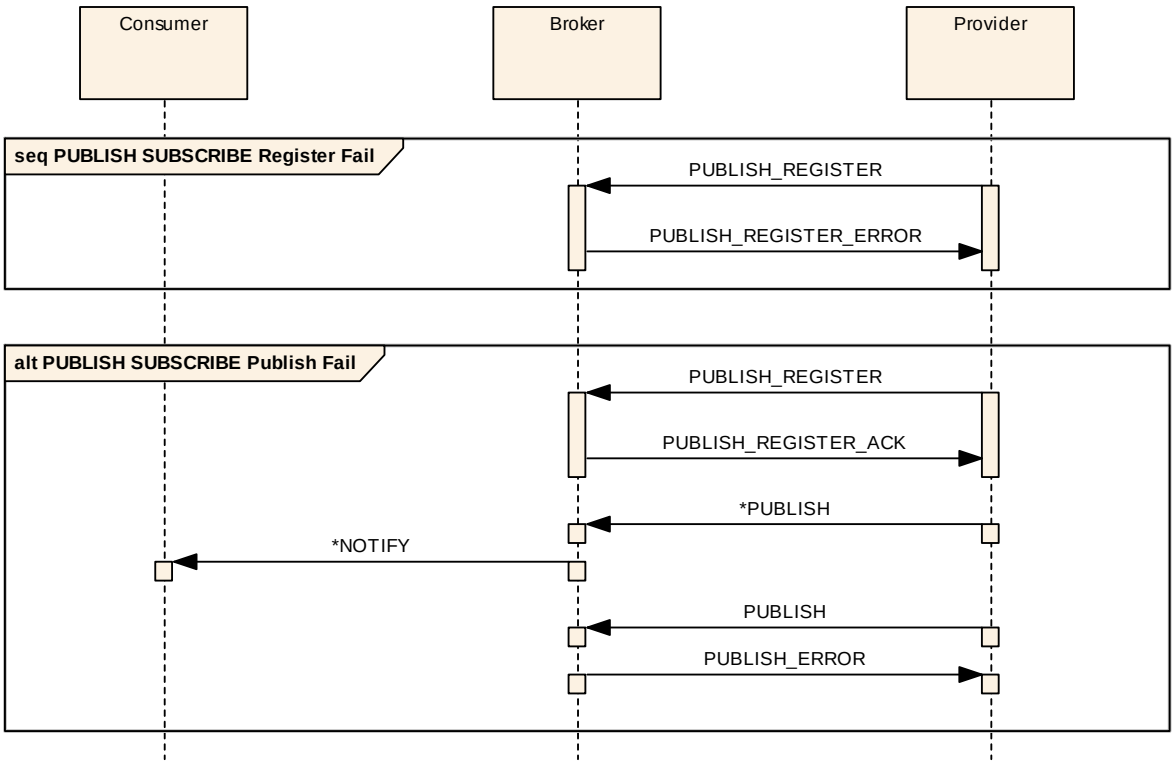

Figure 3-21: PUBLISH-SUBSCRIBE Interaction Pattern Provider Error Sequence

a) The publish register acknowledgement may be replaced with an error message (see 4.4.6) shown in the first sequence in figure 3-21.

b) 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.

c) 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 3-21. The error is sent asynchronously to the provider.

3.5.6.8 Operation Template

The PUBLISH-SUBSCRIBE template is below:

Table 3-24: PUBLISH-SUBSCRIBE Operation Template

Operation Identifier	<<Operation name>>	
Interaction Pattern	PUBLISH-SUBSCRIBE	
Pattern Sequence	Message	Body Type
OUT	PUBLISH/NOTIFY	<<Update type>>

a) The update type shall be the structure that is passed in the provider-to-broker PUBLISH and broker-to-consumer NOTIFY messages.

b) The update type shall extend the abstract Update structure (see 4.4.12).

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	Body Type
IN	REGISTER	Subscription

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

Operation Identifier	publishRegister<< PUBLISH-SUBSCRIBE Operation Name>>	
Interaction Pattern	SUBMIT	
Pattern Sequence	Message	Body Type
IN	PUBLISH_REGISTER	EntityKeyList

Table 3-27: PUBLISH-SUBSCRIBE Publish Operation Template

Operation Identifier	publish<< PUBLISH-SUBSCRIBE Operation Name>>	
Interaction Pattern	SEND	
Pattern Sequence	Message	Body Type
IN	PUBLISH	UpdateList

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

Operation Identifier	publishError<< PUBLISH-SUBSCRIBE Operation Name>>	
Interaction Pattern	SEND	
Pattern Sequence	Message	Body Type
OUT	PUBLISH_ERROR	StandardError

Table 3-29: PUBLISH-SUBSCRIBE Notify Operation Template

Operation Identifier	notify<< PUBLISH-SUBSCRIBE Operation Name>>	
Interaction Pattern	SEND	
Pattern Sequence	Message	Body Type
OUT	NOTIFY	SubscriptionUpdateList

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

Operation Identifier	notifyError<< PUBLISH-SUBSCRIBE Operation Name>>	
Interaction Pattern	SEND	
Pattern Sequence	Message	Body Type
OUT	NOTIFY_ERROR	StandardError

Table 3-31: PUBLISH-SUBSCRIBE Deregister Operation Template

Operation Identifier	deregister<< PUBLISH-SUBSCRIBE Method Name>>	
Interaction Pattern	SUBMIT	
Pattern Sequence	Message	Body Type
IN	DEREGISTER	IdentifierList

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

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

The contents of these messages are fixed and therefore not present in the operation template for PUBLISH-SUBSCRIBE. However, they are directly affected by the template as described below:

a) The consumer register method takes a subscription which is composed of an identifier and a list of entity requests. The identifier shall be used by the broker to uniquely identify the subscription when combined with the consumer URI; this allows a consumer to set up several different concurrent subscriptions.

b) Each subscription shall contain a list of individual entity requests which contain an optional sub-domain, an entity key and an indication of whether updates are required only on change (creation/modification/deletion update types) or for all updates.

c) The provider register message contains a list of entity keys that the provider shall provide updates for.

d) No other entity keys shall be published by that provider.

e) If a provider registers an entity key containing a wildcard for a sub-key then the broker shall allow the provider to publish an update with any value for that sub-key.

f) If a provider publishes an update for an entity that it has not previously registered then the broker shall return an UNKNOWN error in a PUBLISH_ERROR message.

g) The extraInformation field of the StandardError shall be an EntityKeyList that contains the list of entity keys that were unknown.

h) The list of entities allowed to be published by a provider may be replaced by another publish register message.

i) This list shall completely replace the existing list for that provider.

j) The provider-to-broker publish message shall match the provider register message on the session type, session name, area identifier, service identifier, and operation identifier.

k) The provider-to-broker publish message domain shall match exactly, or be a sub-domain of (see 3.5.6.5), the domain of the provider register message.

l) The provider-to-broker publish message shall contain an update list. Each update contains the timestamp of the update, the sourceURI of the provider (used to identify

the source of an update when multiple providers are using a shared broker), an Enumeration denoting if it is considered a new object, an update, a modification or an object deletion by the provider (context specific), and then the service-specific data.

m) The EntityKey of the Update shall not contain the wildcard value.

n) Each update, as listed above, shall be one of four possible types: creation, update, modification, or deletion:

   1) A creation notification shall be used when a new object has been created in the provider (for example, a new parameter is now being published by a service provider).

   2) An update notification shall be used when an object's value has not changed but a new update of it has been generated (for example, a periodic update of a telemetry parameter).

   3) A modification notification shall be used when an object's value has changed from the previously generated notification (for example, a changed telemetry value update).

   4) A deletion notification shall be used when an object is being removed from the provider (for example, an existing parameter is no longer being published by a service provider).

o) The consumer deregister message shall take a list of identifiers of the active subscription lists to cancel.

p) The provider deregister message takes no arguments. It shall completely remove the provider from the set of providers from which updates are permitted.

q) 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.

3.5.6.9 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

3.5.6.10 State Charts

3.5.6.10.1 General

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.

3.5.6.10.2 Consumer Side

Figure 3-22 shows the consumer side state chart for the PUBLISH-SUBSCRIBE pattern:

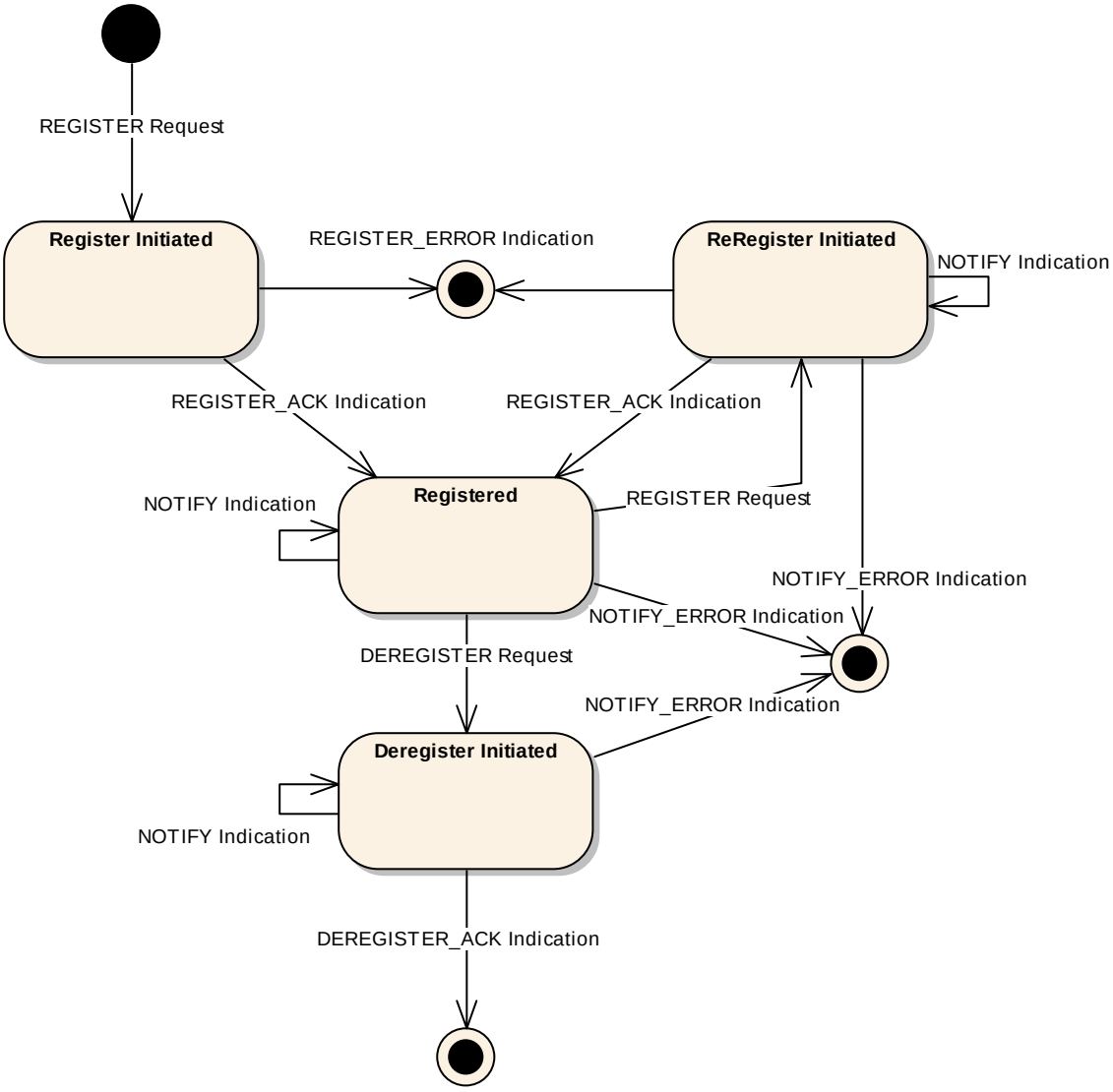

Figure 3-22: PUBLISH-SUBSCRIBE Consumer State Chart

a) The state chart applies to a single subscription for a single consumer, the state chart shall be replicated for each consumer and for each subscription.

b) The transaction identifier of the initial **REGISTER Message** shall be used to identify messages that relate to one PUBLISH-SUBSCRIBE interaction to avoid race conditions.

c) The initial transition is triggered by the primitive **REGISTER Request** and shall lead to the **Register Initiated State**.

d) There are two possible transitions from the **Register Initiated State**:

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

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

e) There are four possible transitions from the **Registered State**:

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

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

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

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

f) There are four possible transitions from the **ReRegistered Initiated State**:

   1) 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.

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

   3) 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 identifier shall be used by the broker to ensure that the reception of the **REGISTER Message** after the **NOTIFY_ERROR Message** was sent shall not cause the restart of the subscription.

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

g) There are three possible transitions from the **Deregister Initiated State**:

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

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

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

3.5.6.10.3 Broker Side (Related to the Consumer)

Figure 3-23 shows the broker to consumer state chart for the PUBLISH-SUBSCRIBE pattern:

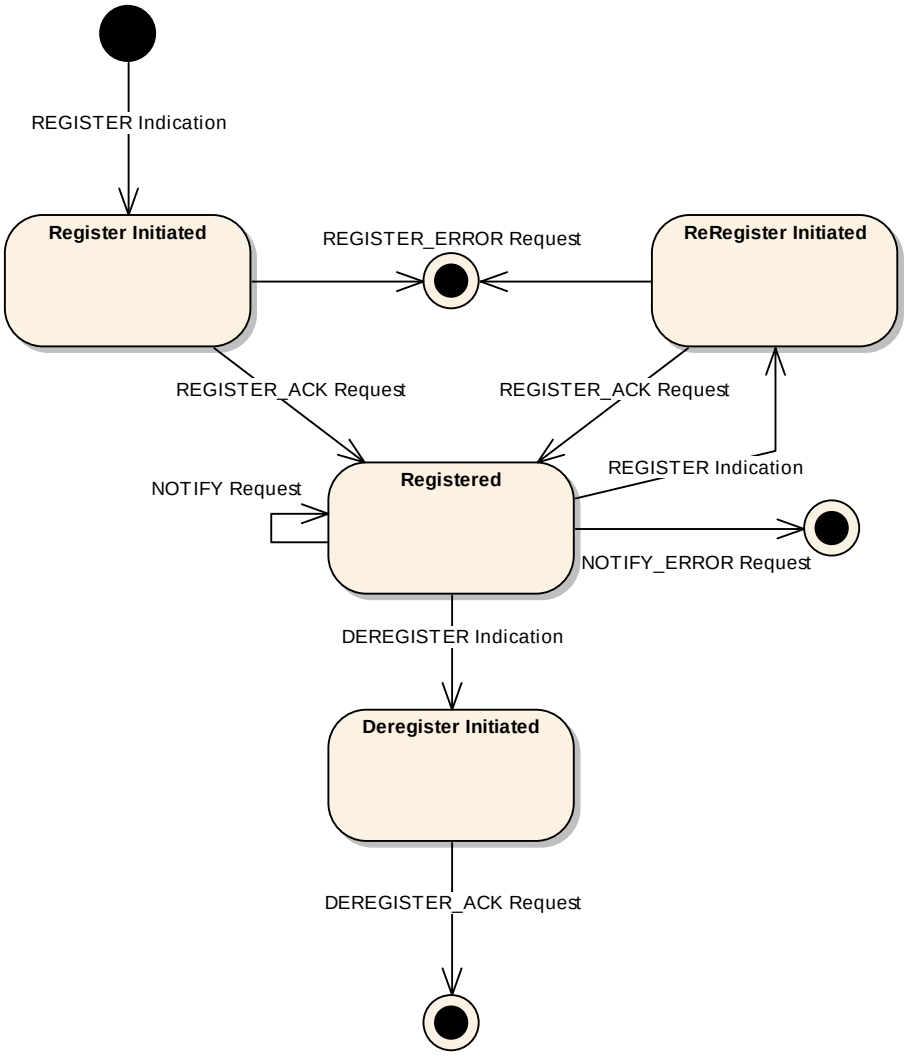

Figure 3-23: PUBLISH-SUBSCRIBE Broker to Consumer State Chart

a) The state chart applies to a single subscription for a single consumer. The state chart shall be replicated for each consumer and for each subscription.

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

c) There are two possible transitions from the **Register Initiated State**:

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

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

d) There are four possible transitions from the **Registered State**;

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

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

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

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

e) There are two possible transitions from the **ReRegister Initiated State**:

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

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

f) 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.

3.5.6.10.4 Provider Side

Figure 3-24 shows the provider side state chart for the PUBLISH-SUBSCRIBE pattern:

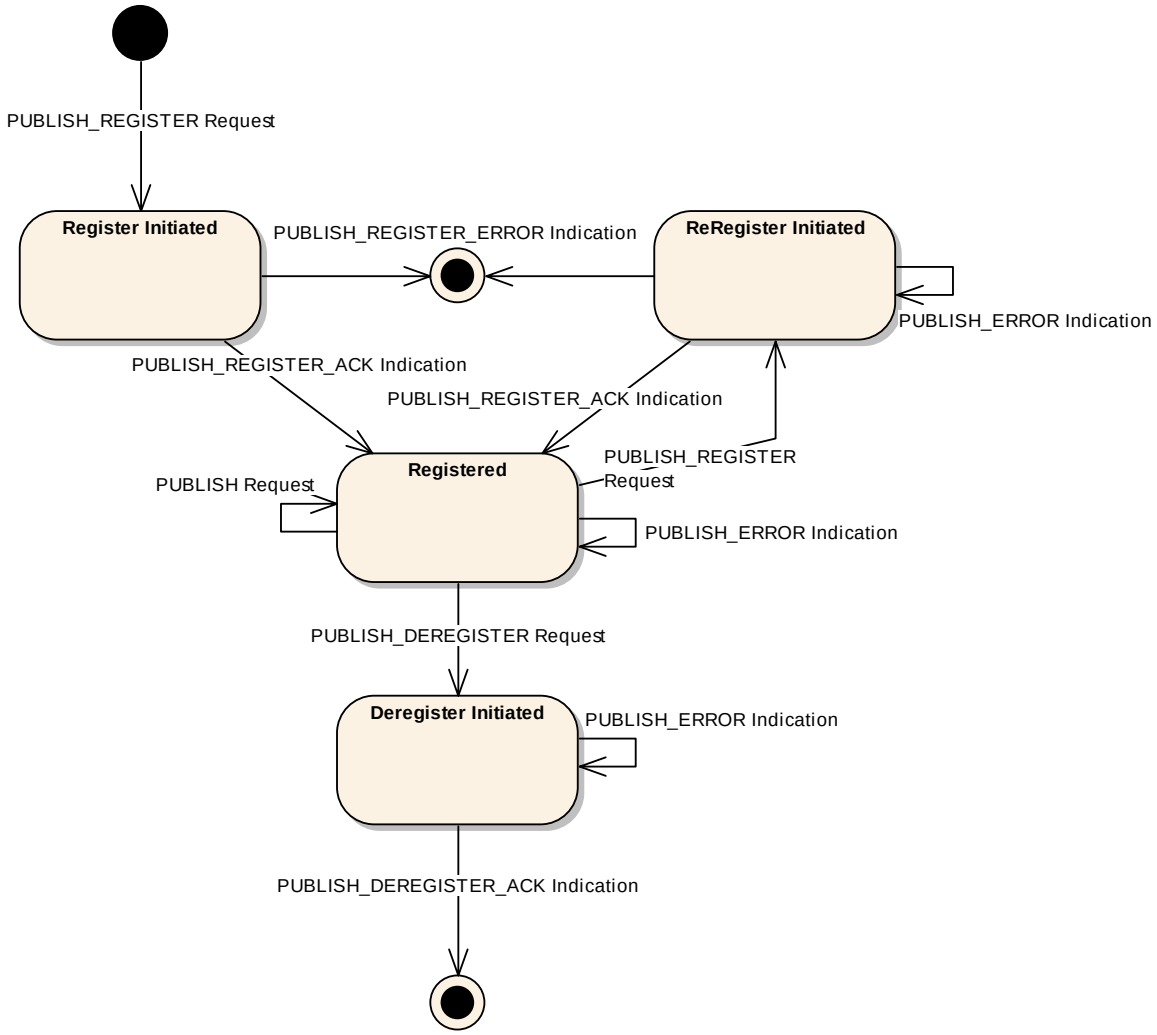

Figure 3-24: PUBLISH-SUBSCRIBE Provider State Chart

a) The state chart applies to a single provider. The state chart shall be replicated for each provider.

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

c) There are two possible transitions from the **Register Initiated State**:

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

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

d) There are four possible transitions from the **Registered State**:

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

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

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

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

e) There are three possible transitions from the **ReRegister Initiated State**:

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

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

3) 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.

f) There are two possible transitions from the **Deregister Initiated State**:

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

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

3.5.6.10.5 Broker Side (Related to the Provider)

Figure 3-25 shows the broker to provider state chart for the PUBLISH-SUBSCRIBE pattern:

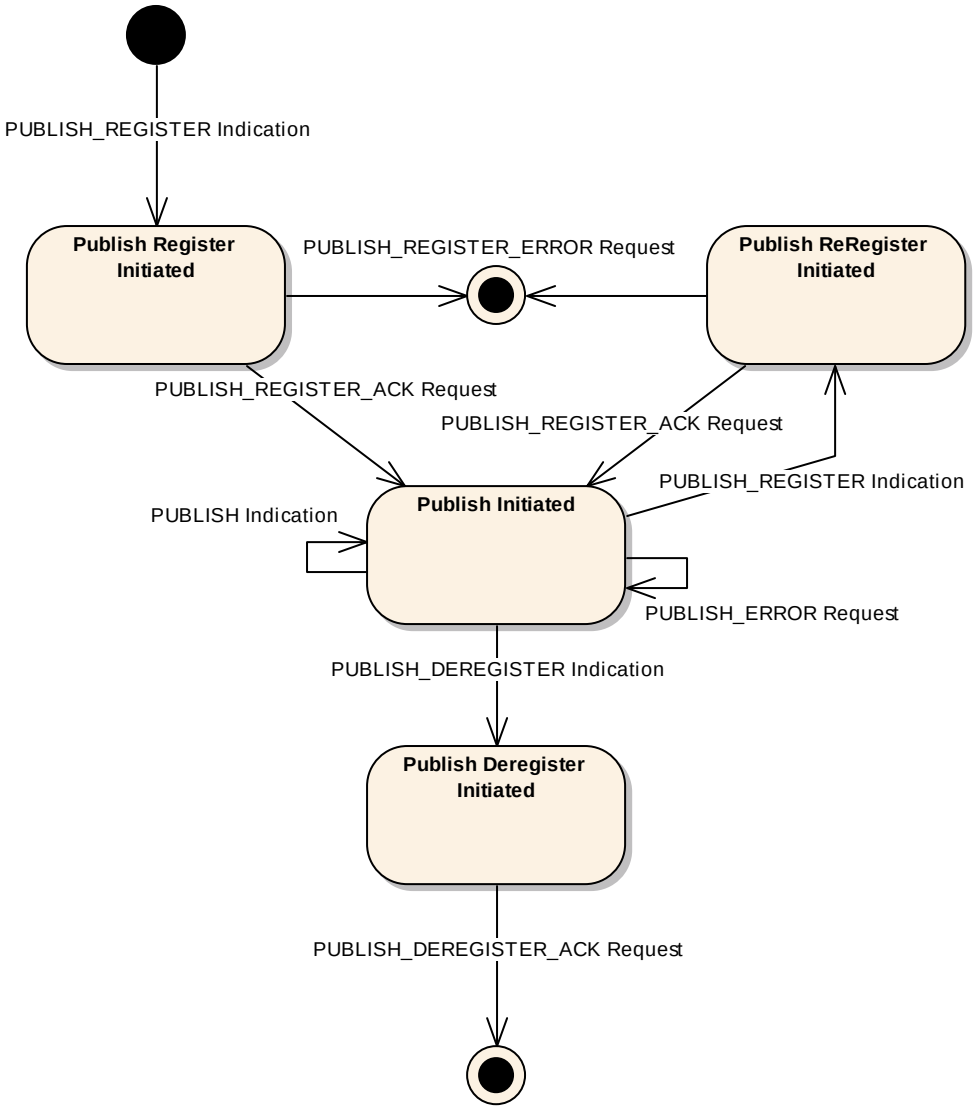

Figure 3-25: PUBLISH-SUBSCRIBE Broker to Provider State Chart

a) The state chart applies to a single provider. The state chart shall be replicated for each provider.

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

c) There are two possible transitions from the **Publish Register Initiated State**:

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

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

d) There are four possible transitions from the **Publish Initiated State**:

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

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

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

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

e) There are two possible transitions from the **Publish ReRegister Initiated State**:

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

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

f) 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.

3.5.6.11 Requests and Indications

3.5.6.11.1 REGISTER

3.5.6.11.1.1 Function

a) 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.

b) 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.

3.5.6.11.1.2 Semantics

REGISTER Request and **REGISTER Indication** shall provide parameters as follows:

(REGISTER Message Header, Subscription, QoS properties)

3.5.6.11.1.3 When Generated

a) 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.

b) 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.

3.5.6.11.1.4 Effect on Reception

a) 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.

b) The consumer MAL shall enter the **Register Initiated State** in this case.

c) 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.

d) The consumer MAL shall enter the **ReRegister Initiated State** in this case.

e) 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.

f) 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.

g) The broker shall start processing the subscription at this point.

3.5.6.11.1.5 Message Header

a) For the **REGISTER Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-34.

b) 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
URI From	Consumer URI
Authentication Id	Consumer Authentication Identifier
URI To	Broker URI
Interaction Type	PUBSUB
Interaction Stage	1
Transaction Id	Provided by consumer MAL

3.5.6.11.2 REGISTER_ACK

3.5.6.11.2.1 Function

a) The **REGISTER_ACK Request** primitive shall be used by the broker to acknowledge a PUBSUB Interaction subscription.

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

3.5.6.11.2.2 Semantics

REGISTER_ACK Request and **REGISTER_ACK Indication** shall provide parameters as follows:

(REGISTER_ACK Message Header, QoS properties)

3.5.6.11.2.3 When Generated

a) 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.

b) 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.

3.5.6.11.2.4 Effect on Reception

a) 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.

b) A broker MAL in the **Register Initiated State** or **ReRegister Initiated State** shall enter the **Registered State**.

c) 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.

3.5.6.11.2.5 Message Header

For the **REGISTER_ACK Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-35.

Table 3-35: REGISTER_ACK Message Header Fields

Field	Value
URI From	Broker URI
Authentication Id	Broker Authentication Identifier
URI To	Consumer URI
QoSlevel	QoS level from first REGISTER message
Priority	Priority from first REGISTER message
Interaction Type	PUBSUB
Interaction Stage	2
Transaction Id	Transaction Id from initial message

3.5.6.11.3 REGISTER_ERROR

3.5.6.11.3.1 Function

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

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

3.5.6.11.3.2 Semantics

REGISTER_ERROR Request and **REGISTER_ERROR Indication** shall provide parameters as follows:

(REGISTER_ERROR Message Header, StandardError, QoS properties)

3.5.6.11.3.3 When Generated

a) 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.

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

   1) Too many subscriptions for what the broker can support shall cause a TOO_MANY error to be sent. The extraInformation field of the error contains an Integer which provides the maximum number of subscriptions supported.

   2) Shutdown, the broker is shutting down. A SHUTDOWN error message shall be returned.

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

   4) 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.

c) 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:

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

   2) an error raised by the local communication layer.

3.5.6.11.3.4 Effect on Reception

a) 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.

b) The pattern shall end at this point for the broker.

c) 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.

3.5.6.11.3.5 Message Header

a) 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.

b) The contents of the StandardError structure shall immediately follow the message header.

3.5.6.11.4 PUBLISH_REGISTER

3.5.6.11.4.1 Function

a) The **PUBLISH_REGISTER Request** primitive shall be used by the provider application to initiate a PUBSUB Interaction with the list of entities being published or to update an existing list.

b) 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.

3.5.6.11.4.2 Semantics

PUBLISH_REGISTER Request and **PUBLISH_REGISTER Indication** shall provide parameters as follows:

(PUBLISH_REGISTER Message Header, EntityKeyList, QoS properties)

3.5.6.11.4.3 When Generated

a) 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 publish list.

b) 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.

3.5.6.11.4.4 Effect on Reception

a) 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.

b) The provider MAL shall enter the **Register Initiated State** in this case.

c) 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 update the publishing list by transmitting a **PUBLISH_REGISTER Message** to the broker.

d) The provider MAL shall enter the **ReRegister Initiated State** in this case.

e) 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.

f) The broker shall store the publish list of the provider or update the existing list from the same provider.

3.5.6.11.4.5 Message Header

a) For the **PUBLISH_REGISTER Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-36.

b) The contents of the EntityKeyList, as specified in the operation template, shall immediately follow the message header.

Table 3-36: PUBLISH_REGISTER Message Header Fields

Field	Value
URI From	Provider URI
Authentication Id	Provider Authentication Identifier
URI To	Broker URI
Interaction Type	PUBSUB
Interaction Stage	3
Transaction Id	Provided by provider MAL

3.5.6.11.5 PUBLISH_REGISTER_ACK

3.5.6.11.5.1 Function

a) The **PUBLISH_REGISTER_ACK Request** primitive shall be used by the broker to acknowledge a PUBSUB Interaction publish list from a provider.

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

3.5.6.11.5.2 Semantics

PUBLISH_REGISTER_ACK Request and **PUBLISH_REGISTER_ACK Indication** shall provide parameters as follows:

(PUBLISH_REGISTER Message Header, QoS properties)

3.5.6.11.5.3 When Generated

a) 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.

b) 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.

3.5.6.11.5.4 Effect on Reception

a) 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.

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

c) 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.

3.5.6.11.5.5 Message Header

For the **PUBLISH_REGISTER_ACK Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-37.

Table 3-37: PUBLISH_REGISTER_ACK Message Header Fields

Field	Value
URI From	Broker URI
Authentication Id	Broker Authentication Identifier
URI To	Provider URI
QoSlevel	QoS level from first PUBLISH_REGISTER message
Priority	Priority from first PUBLISH_REGISTER message
Interaction Type	PUBSUB
Interaction Stage	4
Transaction Id	Transaction Id from provider register message

3.5.6.11.6 PUBLISH_REGISTER_ERROR

3.5.6.11.6.1 Function

a) 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.

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

3.5.6.11.6.2 Semantics

PUBLISH_REGISTER_ERROR Request and **PUBLISH_REGISTER_ERROR Indication** shall provide parameters as follows:

(PUBLISH_REGISTER_ERROR Message Header, StandardError, QoS properties)

3.5.6.11.6.3 When Generated

a) 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.

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

   1) Too many providers for what the broker can support shall cause a TOO_MANY error to be sent. The extraInformation field of the error contains an Integer which provides the maximum number of providers supported.

   2) Shutdown, the broker is shutting down. A SHUTDOWN error message shall be returned.

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

c) 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:

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

   2) an error raised by the local communication layer.

3.5.6.11.6.4 Effect on Reception

a) 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.

b) The pattern shall end at this point for the broker.

c) 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.

3.5.6.11.6.5 Message Header

a) 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.

b) The contents of the StandardError structure shall immediately follow the message header.

3.5.6.11.7 PUBLISH

3.5.6.11.7.1 Function

a) The **PUBLISH Request** primitive shall be used by the provider to publish a list of entity updates.

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

3.5.6.11.7.2 Semantics

PUBLISH Request and **PUBLISH Indication** shall provide parameters as follows:

(PUBLISH Message Header, UpdateList, QoS properties)

3.5.6.11.7.3 When Generated

a) A **PUBLISH Request** shall be used by the provider with the provider MAL in the **Registered State** to publish a list of entity updates.

b) 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.

3.5.6.11.7.4 Effect on Reception

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

b) 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.

c) The broker application shall match the Updates in the list to the subscriptions of registered consumers that are in the **Registered State** using the match rules in 3.5.6.5.

d) The matched updates shall be transmitted to the relevant consumers by the broker using a **NOTIFY Request**.

3.5.6.11.7.5 Message Header

a) For the **PUBLISH Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-38.

b) The contents of the UpdateList, as specified in the operation template, shall immediately follow the message header.

Table 3-38: PUBLISH Message Header Fields

Field	Value
URI From	Provider URI
Authentication Id	Provider Authentication Identifier
URI To	Broker URI
Interaction Type	PUBSUB
Interaction Stage	5
Transaction Id	Transaction Id from provider register message

3.5.6.11.8 PUBLISH_ERROR

3.5.6.11.8.1 Function

a) The **PUBLISH_ERROR Request** primitive shall be used by the broker to reject a publish request from a provider.

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

3.5.6.11.8.2 Semantics

PUBLISH_ERROR Request and **PUBLISH_ERROR Indication** shall provide parameters as follows:

(PUBLISH_ERROR Message Header, StandardError, QoS properties)

3.5.6.11.8.3 When Generated

a) A **PUBLISH_ERROR Request** shall be used by the broker with the broker MAL in the **Publish Initiated State** to reject a publish request.

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

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

   2) Unknown Entity: a provider has attempted to publish a previously unregistered Entity. An UNKNOWN error shall be returned in this case; the extraInformation field of the error contains an EntityKey list which contains the list of unknown EntityKeys.

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

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

c) 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**.

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

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

   2) an error raised by the local communication layer.

3.5.6.11.8.4 Effect on Reception

a) 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.

b) 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.

3.5.6.11.8.5 Message Header

a) 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.

b) In the case where an error is being return without a previous PUBLISH_REGISTER message, the QoSlevel, Priority, and Transaction Identifier shall be taken from the PUBLISH message.

c) The contents of the StandardError structure shall immediately follow the message header.

Table 3-39: PUBLISH_ERROR Message Header Fields

Field	Value
URI From	Broker URI
Authentication Id	Broker Authentication Identifier
URI To	Provider URI
QoSlevel	QoS level from first PUBLISH_REGISTER message
Priority	Priority from first PUBLISH_REGISTER message
Is Error Message	TRUE

3.5.6.11.9 NOTIFY

3.5.6.11.9.1 Function

a) The **NOTIFY Request** primitive shall be used by the broker to transmit a list of entity updates to a consumer.

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

3.5.6.11.9.2 Semantics

NOTIFY Request and **NOTIFY Indication** shall provide parameters as follows:

(NOTIFY Message Header, SubscriptionUpdateList, QoS properties)

3.5.6.11.9.3 When Generated

a) A **NOTIFY Request** shall be used by the broker with the broker MAL in the **Registered State** to transmit a list of entity updates from a provider **PUBLISH Message** to a consumer.

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

3.5.6.11.9.4 Effect on Reception

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

b) 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.

3.5.6.11.9.5 Message Header

a) For the **NOTIFY Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-40.

b) The contents of the SubscriptionUpdateList, as specified in the operation template, shall immediately follow the message header.

c) The transaction identifier 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 identifier of the first request shall be used.

Table 3-40: NOTIFY Message Header Fields

Field	Value
URI From	Broker URI
Authentication Id	Broker Authentication Identifier
URI To	Consumer URI
QoSlevel	QoS level from first REGISTER message
Priority	Priority from first REGISTER message
Interaction Type	PUBSUB
Interaction Stage	6
Transaction Id	Transaction Id from consumer register message

3.5.6.11.10 NOTIFY_ERROR

3.5.6.11.10.1 Function

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

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

3.5.6.11.10.2 Semantics

NOTIFY_ERROR Request and **NOTIFY_ERROR Indication** shall provide parameters as follows:

(NOTIFY_ERROR Message Header, StandardError, QoS properties)

3.5.6.11.10.3 When Generated

a) 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.

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

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

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

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

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

   2) an error raised by the local communication layer.

3.5.6.11.10.4 Effect on Reception

a) 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.

b) The pattern shall end at this point for the broker.

c) 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.

3.5.6.11.10.5 Message Header

a) 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.

b) The contents of the StandardError structure shall immediately follow the message header.

3.5.6.11.11 DEREGISTER

3.5.6.11.11.1 Function

a) The **DEREGISTER Request** primitive shall be used by the consumer to end subscriptions for the specified subscription identifiers.

b) 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.

3.5.6.11.11.2 Semantics

DEREGISTER Request and **DEREGISTER Indication** shall provide parameters as follows:

(DEREGISTER Message Header, IdentifierList, QoS properties)

3.5.6.11.11.3 When Generated

a) 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.

b) 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.

3.5.6.11.11.4 Effect on Reception

a) 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.

b) The consumer MAL shall enter the **Deregister Initiated State** for each identified subscription.

c) 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.

d) The broker shall then remove the specified subscriptions of the consumer.

3.5.6.11.11.5 Message Header

a) For the **DEREGISTER Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-41.

b) The contents of the subscription IdentifierList, as specified in the operation template, shall immediately follow the message header.

Table 3-41: DEREGISTER Message Header Fields

Field	Value
URI From	Consumer URI
Authentication Id	Consumer Authentication Identifier
URI To	Broker URI
Interaction Type	PUBSUB
Interaction Stage	7
Transaction Id	Provided by consumer MAL

3.5.6.11.12 DEREGISTER_ACK

3.5.6.11.12.1 Function

a) The **DEREGISTER_ACK Request** primitive shall be used by the broker to acknowledge the termination of a PUBSUB Interaction subscription by the consumer.

b) The **DEREGISTER_ACK Indication** primitive shall be used by the consumer MAL to deliver a **DEREGISTER_ACK Message** to a consumer.

3.5.6.11.12.2 Semantics

DEREGISTER_ACK Request and **DEREGISTER_ACK Indication** shall provide parameters as follows:

(DEREGISTER_ACK Message Header, QoS properties)

3.5.6.11.12.3 When Generated

a) 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.

b) 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.

3.5.6.11.12.4 Effect on Reception

a) 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.

b) The interaction shall end at this point for the broker for each of the identified subscriptions.

c) 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.

d) The interaction shall end here for the consumer for the set of subscriptions in the initiating **DEREGISTER Message**.

3.5.6.11.12.5 Message Header

For the **DEREGISTER_ACK Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-42.

Table 3-42: DEREGISTER_ACK Message Header Fields

Field	Value
URI From	Broker URI
Authentication Id	Broker Authentication Identifier
URI To	Consumer URI
QoSlevel	QoS level from first REGISTER message
Priority	Priority from first REGISTER message
Interaction Type	PUBSUB
Interaction Stage	8
Transaction Id	Transaction Id from initial message

3.5.6.11.13 PUBLISH_DEREGISTER

3.5.6.11.13.1Function

a) The **PUBLISH_DEREGISTER Request** primitive shall be used by the provider to end a PUBSUB Interaction.

b) 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.

3.5.6.11.13.2 Semantics

PUBLISH_DEREGISTER Request and **PUBLISH_DEREGISTER Indication** shall provide parameters as follows:

(PUBLISH_DEREGISTER Message Header, QoS properties)

3.5.6.11.13.3 When Generated

a) A **PUBLISH_DEREGISTER Request** shall be used by the provider application with the provider MAL in the **Registered State** to end the interaction.

b) 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.

3.5.6.11.13.4 Effect on Reception

a) 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.

b) The provider MAL shall enter the **Deregister Initiated State**.

c) 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**.

d) The broker shall then remove the provider from the list of allowed publishers.

3.5.6.11.13.5 Message Header

For the **PUBLISH_DEREGISTER Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-43.

Table 3-43: PUBLISH_DEREGISTER Message Header Fields

Field	Value
URI From	Provider URI
Authentication Id	Provider Authentication Identifier
URI To	Broker URI
Interaction Type	PUBSUB
Interaction Stage	9
Transaction Id	Provided by provider MAL

3.5.6.11.14 PUBLISH_DEREGISTER_ACK

3.5.6.11.14.1 Function

a) The **PUBLISH_DEREGISTER_ACK Request** primitive shall be used by the broker to acknowledge the termination of a PUBSUB Interaction by the provider.

b) The **PUBLISH_DEREGISTER_ACK Indication** primitive shall be used by the provider MAL to deliver a **PUBLISH_DEREGISTER_ACK Message** to a provider.

3.5.6.11.14.2 Semantics

PUBLISH_DEREGISTER_ACK Request and **PUBLISH_DEREGISTER_ACK Indication** shall provide parameters as follows:

(PUBLISH_DEREGISTER_ACK Message Header, QoS properties)

3.5.6.11.14.3 When Generated

a) 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.

b) 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.

3.5.6.11.14.4 Effect on Reception

a) 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.

b) The interaction shall end at this point for the broker.

c) 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.

d) The interaction shall end here for the provider.

3.5.6.11.14.5 Message Header

For the **PUBLISH_DEREGISTER_ACK Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-44.

Table 3-44: PUBLISH_DEREGISTER_ACK Message Header Fields

Field	Value
URI From	Broker URI
Authentication Id	Broker Authentication Identifier
URI To	Provider URI
QoSlevel	QoS level from first PUBLISH_REGISTER message
Priority	Priority from first PUBLISH_REGISTER message
Interaction Type	PUBSUB
Interaction Stage	10
Transaction Id	Transaction Id from provider deregister message

3.5.6.12 Example

The following example shows a simple example service that contains a single PUBLISH-SUBSCRIBE operation:

Operation Identifier	testPubSub	
Interaction Pattern	PUBLISH-SUBSCRIBE	
Pattern Sequence	Message	Body Type
OUT	PUBLISH/NOTIFY	TestNotify

The TestNotify structure is defined below:

Structure Name	TestNotify	
Extends	Update	
Short form	Example Only	
Field	Type	Comment
time	Time	Example Time item
value	Integer	Example Integer item

To register for this PUBLISH-SUBSCRIBE pattern, a consumer shall send the following message:

Message Header															
URI From	Auth Id	URI To	Timestamp	QoS	Priority	Domain	Zone	Session	Interaction	Stage	Trans Id	Area	Service	Operation	Version	Is Error
Consumer	Op A	Broker		BEST	1	A.B.C	GROUND	LIVE	PUBSUB	1	1238	Example	Example	testPubSub	1	FALSE

Subscription															
	EntityRequestList														
Subscription Id	List count	EntityRequest							EntityRequest						
		EntityKeyList						Only On Change	EntityKeyList						Only On Change
		List count	EntityKey						List count	EntityKey					
			1st	2nd	3rd	4th				1st	2nd	3rd	4th		
Sub 123	2	1	A	B	*	*	False		1	A	C	*	*	True	

It contains a standard pattern body for the register message. This shall result in the following acknowledgement message being sent:

Message Header															
URI From	Auth Id	URI To	Timestamp	QoS	Priority	Domain	Zone	Session	Interaction	Stage	Trans Id	Area	Service	Operation	Version	Is Error
Broker	BR Q	Consumer		BEST	1	A.B.C	GROUND	LIVE	PUBSUB	2	1238	Example	Example	testPubSub	1	FALSE

To publish an update on this topic a previously registered provider would send the following message to the message broker. In this example two updates are being generated at the same time:

Message Header																
URI From	Auth Id	URI To	Timestamp	QoS	Priority	Domain	Zone	Session	Interaction	Stage	Trans Id	Area	Service	Operation	Version	Is Error
Provider	SC X	Broker		BEST	1	A.B.C	GROUND	LIVE	PUBSUB	5		Example	Example	testPubSub	1	FALSE

UpdateList																	
List count	Test Notify								Test Notify								
	Time stamp	Source URI	Update Type	EntityKey				Time	Value	Time stamp	Source URI	Update Type	EntityKey				Time	Value
				1st	2nd	3rd	4th						1st	2nd	3rd	4th		
2		SC X	Update	A	B	1	2	Today, 09:30	1234		SC X	Update	A	C	1	2	Today, 09:30	8888

When an update is required the following notify message will be sent from the message broker to the consumer:

Message Header																
URI From	Auth Id	URI To	Timestamp	QoS	Priority	Domain	Zone	Session	Interaction	Stage	Trans Id	Area	Service	Operation	Version	Is Error
Broker	BR Q	Consumer		BEST	1	A.B.C	GROUND	LIVE	PUBSUB	6		Example	Example	testPubSub	1	FALSE

SubscriptionUpdateList												
List count	SubscriptionUpdate											
	Subscription Id	UpdateList										
		List count	Test Notify									
			Time stamp	Source URI	Update Type	EntityKey				Time	Time stamp	
						1st	2nd	3rd	4th		Value	
1	Sub 123	1		SC X	Update	A	B	1	2	Today, 09:30	1234	

No acknowledgment of the notify message shall be sent by the consumer.

When the consumer wants to deregister, the following message would be sent:

Message Header																	IdentifierList	
URI From	Auth Id	URI To	Timestamp	QoS	Priority	Domain	Zone	Session	Interaction	Stage	Trans Id	Area	Service	Operation	Version	Is Error	List count	Id
Consumer	Op A	Broker		BEST	1	A.B.C	GROUND	LIVE	PUBSUB	7	1238	Example	Example	testPubSub	1	FALSE	1	Sub 123

Which would result in the following acknowledgment being received:

Message Header																
URI From	Auth Id	URI To	Timestamp	QoS	Priority	Domain	Zone	Session	Interaction	Stage	Trans Id	Area	Service	Operation	Version	Is Error
Broker	BR Q	Consumer		BEST	1	A.B.C	GROUND	LIVE	PUBSUB	8	1238	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.

3.6 ACCESS CONTROL INTERFACE

3.6.1 CHECK MESSAGE INTERACTION

3.6.1.1 Overview

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 (figure 3-26). No acknowledgement other than the response is sent.

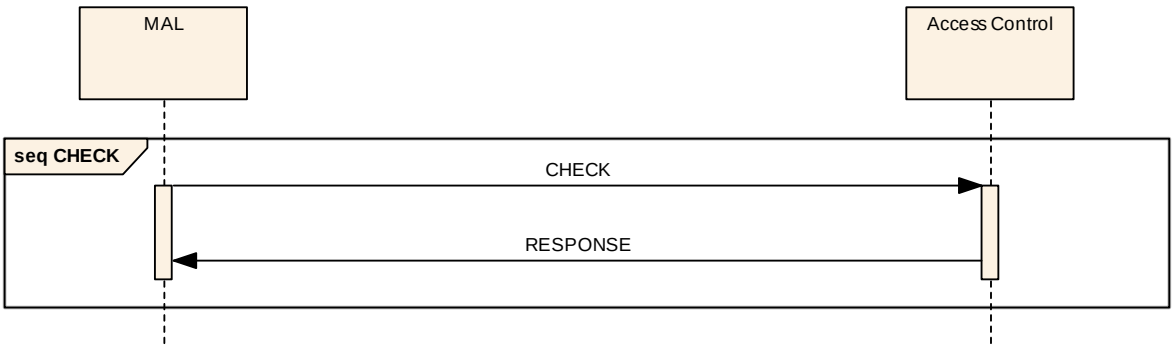

Figure 3-26: CHECK Access Control Pattern Message Sequence

3.6.1.2 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.

3.6.1.3 Usage

a) 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.

b) A compliant MAL implementation shall follow the sequences defined in sections 4 and 5 of reference [1] for interacting with an access control component.

c) A broker in a Publish Subscribe pattern shall also submit any NOTIFY and PUBLISH Messages to its Access Control component.

d) 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.

3.6.1.4 Error Handling

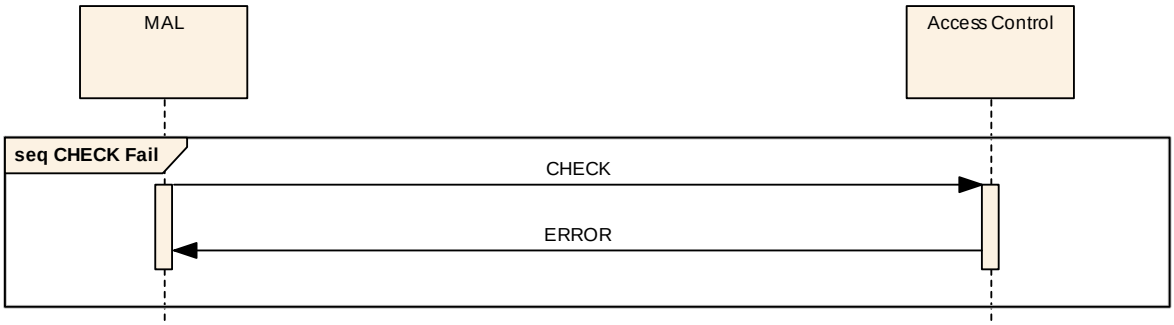

Figure 3-27: CHECK Access Control Pattern Error Sequence

a) The response shall be replaced with an error message (see 4.4.6) if an error occurs during the processing of the operation (figure 3-27).

b) Either the response or the error message shall be returned but never both.

3.6.1.5 Operation Template

The CHECK pattern template is below:

Table 3-45: CHECK Operation Template

Interaction Pattern	CHECK	
Pattern Sequence	Message	Body Type
IN	CHECK	MAL message
OUT	RESPONSE	MAL message

3.6.1.6 Primitives

The pattern is not an abstract interface that provides interoperability. However, it is a function that is required by the MAL.

Table 3-46: CHECK Primitive List

Primitive
CHECK
RESPONSE
ERROR

3.6.1.7 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.

3.6.1.8 Requests and Indications

3.6.1.8.1 CHECK

3.6.1.8.1.1 Function

The **CHECK Request** primitive shall be used by the MAL to initiate a CHECK Interaction.

3.6.1.8.1.2 Semantics

CHECK Request shall provide parameters as follows:

(MAL Message, QoS properties)

3.6.1.8.1.3 When Generated

a) **CHECK Request** shall be generated by the MAL after reception of a message from either the application or from a transport.

b) It is implementation and deployment specific what checks an implementation of the Access Control component shall perform upon reception of a **CHECK Request**.

3.6.1.8.2 RESPONSE

3.6.1.8.2.1 Function

The **RESPONSE Indication** primitive shall be used by the Access Control component to deliver a **RESPONSE Message** to the MAL.

3.6.1.8.2.2 Semantics

RESPONSE Indication shall provide parameters as follows:

(MAL Message, QoS properties)

3.6.1.8.2.3 When Generated

RESPONSE Indication shall be generated upon reception of a **RESPONSE Message** from the Access Control component.

3.6.1.8.2.4 Effect on Reception

a) On reception of a **RESPONSE Indication** the MAL shall end the interaction pattern with success.

b) The returned message shall then be used by the MAL from that point onwards.

3.6.1.8.3 ERROR

3.6.1.8.3.1 Function

The **ERROR Indication** primitive shall be used by the Access Control component to end a CHECK Interaction with an error.

3.6.1.8.3.2 Semantics

ERROR Indication shall provide parameters as follows:

(StandardError, QoS properties)

3.6.1.8.3.3 When Generated

a) An **ERROR Indication** shall be generated upon reception of an **ERROR Message** from the Access Control component.

b) The Access Control component shall return AUTHENTICATION_FAIL if the supplied message fails authentication checks. These checks are implementation and deployment specific.

c) The Access Control component shall return AUTHORISATION_FAIL if the authenticated supplied message fails authorisation checks. These checks are implementation and deployment specific.

3.6.1.8.3.4 Effect on Reception

a) On reception of an **ERROR Indication** the MAL shall end the interaction with an error.

b) 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.

c) The behaviour of the MAL from this point onwards is defined in reference [1].

3.7 TRANSPORT INTERFACE

3.7.1 GENERAL

This subsection defines the abstract interface that a Transport layer provides to the MAL. It specifies what facilities must be made available to a compliant MAL and also the required behaviour of the Transport.

The specification of this for a particular technology is called a Transport Specification. It defines the mapping from the abstract MAL data structures into a specific and unambiguous encoding of the messages for that specific data transport. (For further information on this, see reference [1].)

a) A Transport Specification shall list, for each supported QoS, the MAL errors it may raise.

b) A Transport Specification shall define, for each supported QoS, the conditions which trigger the raising of the listed MAL errors.

c) A Transport Specification shall define, for each supported QoS, the QoS properties it supports.

d) A Transport Specification shall define, for each supported QoS, the effect the QoS properties have on its behaviour.

e) A Transport Specification shall define, for each supported QoS, the message timeout behaviour.

f) If Transport Specification supports the QUEUED QoS, then it shall define its behaviour in terms of message persistence and purging.

3.7.2 SUPPORTEDQOS INTERACTION

3.7.2.1 Overview

The SUPPORTEDQOS pattern is similar to the REQUEST Interaction Pattern. No acknowledgement other than the response is sent (figure 3-28).

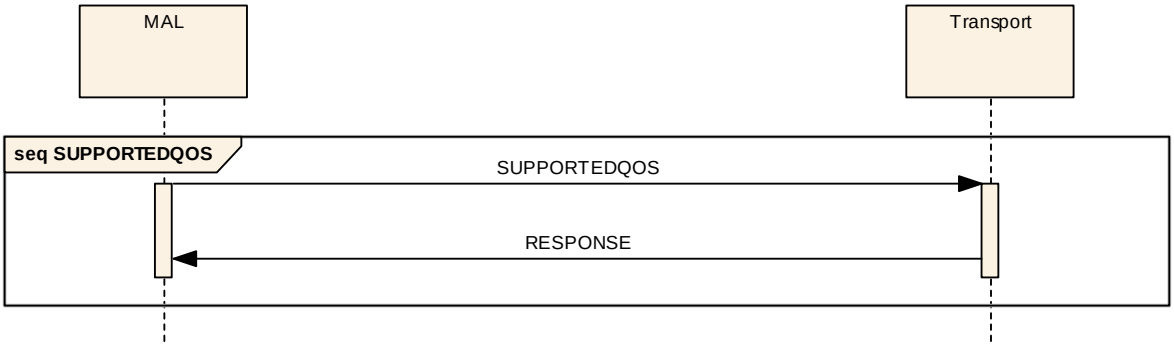

Figure 3-28: SUPPORTEDQOS Transport Pattern Message Sequence

3.7.2.2 Description

The SUPPORTEDQOS pattern provides a simple request/response message exchange that is used by a MAL implementation to determine which QoS levels are supported by a specific Transport layer implementation.

NOTE – It is not expected that this pattern will be implemented via a message, as it is expected to be implemented as a local API used by the MAL to access the Transport layer. It is shown here as a pattern for consistency.

3.7.2.3 Usage

The SUPPORTEDQOS pattern shall be used only by the MAL for the checking of supported QoS levels. It is not used by any other component.

3.7.2.4 Error Handling

No Errors shall be raised.

3.7.2.5 Operation Template

The SUPPORTEDQOS pattern template is below:

Table 3-47: SUPPORTEDQOS Operation Template

Interaction Pattern	SUPPORTEDQOS	
Pattern Sequence	Message	Body Type
IN	SUPPORTEDQOS	QoSLevel
OUT	RESPONSE	Boolean

3.7.2.6 Primitives

The pattern is not an abstract interface that provides interoperability. However, it is a function that is required by the MAL.

Table 3-48: SUPPORTEDQOS Primitive List

Primitive
SUPPORTEDQOS
RESPONSE

3.7.2.7 State Charts

No state charts are provided for this pattern. The MAL shall initiate the pattern using the initial primitive and shall block until a response is received.

3.7.2.8 Requests and Indications

3.7.2.8.1 SUPPORTEDQOS

3.7.2.8.1.1 Function

The **SUPPORTEDQOS Request** primitive shall be used by the MAL to determine if a Transport supports a specific QoS level.

3.7.2.8.1.2 Semantics

SUPPORTEDQOS Request shall provide parameters as follows:

(QoSLevel)

3.7.2.8.1.3 When Generated

SUPPORTEDQOS Request shall be generated by the MAL when first interacting with a specific Transport.

3.7.2.8.1.4 Effect on Reception

Reception of a **SUPPORTEDQOS Request** shall result in a return of a **RESPONSE Indication**.

3.7.2.8.2 RESPONSE

3.7.2.8.2.1 Function

a) The **RESPONSE Indication** primitive shall be used by a Transport layer to deliver the response to a **SUPPORTEDQOS Request** to the MAL.

b) If the Transport supports the QoS level it shall return ‘True’ in the indication, ‘False’ otherwise.

3.7.2.8.2.2 Semantics

RESPONSE Indication shall provide parameters as follows:

(Boolean)

3.7.2.8.2.3 When Generated

RESPONSE Indication shall be generated by the Transport layer in response to a **SUPPORTEDQOS Request**.

3.7.2.8.2.4 Effect on Reception

The effect on reception of a **RESPONSE Indication** by the MAL is MAL implementation specific.

3.7.3 SUPPORTEDIP INTERACTION

3.7.3.1 Overview

The SUPPORTEDIP pattern is similar to the REQUEST Interaction Pattern. No acknowledgement other than the response is sent (figure 3-29).

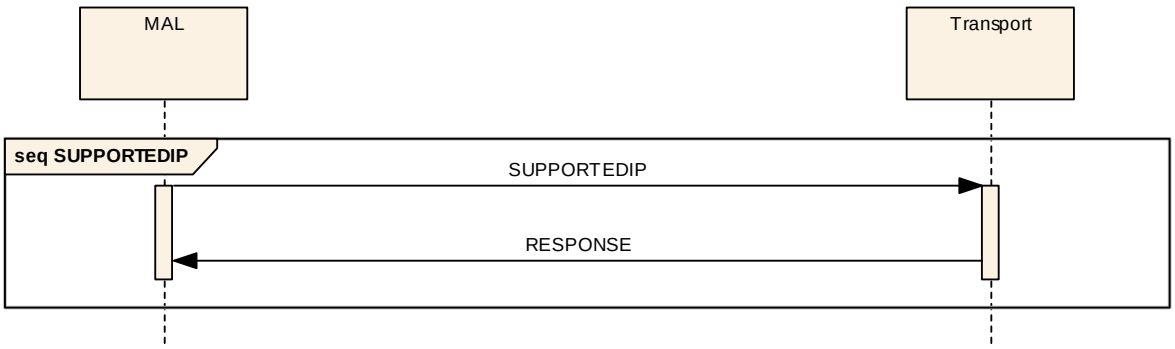

Figure 3-29: SUPPORTEDIP Transport Pattern Message Sequence

3.7.3.2 Description

The SUPPORTEDIP pattern provides a simple request/response message exchange that is used by a MAL implementation to determine which MAL Interaction Patterns are supported by a specific Transport layer implementation.

NOTE – It is not expected that this pattern will be implemented via a message, as it is expected to be implemented as a local API used by the MAL to access the Transport layer. It is shown here as a pattern for consistency.

3.7.3.3 Usage

The SUPPORTEDIP pattern is only used by the MAL for the checking of supported interaction patterns. It is not expected to be used by any other component.

3.7.3.4 Error Handling

No Errors shall be raised.

3.7.3.5 Operation Template

The SUPPORTEDIP pattern template is below:

Table 3-49: SUPPORTEDIP Operation Template

Interaction Pattern	SUPPORTEDIP	
Pattern Sequence	Message	Body Type
IN	SUPPORTEDIP	InteractionType
OUT	RESPONSE	Boolean

3.7.3.6 Primitives

The pattern is not an abstract interface that provides interoperability. However, it is a function that is required by the MAL.

Table 3-50: SUPPORTEDIP Primitive List

Primitive
SUPPORTEDIP
RESPONSE

3.7.3.7 State Charts

No state charts are provided for this pattern. The MAL shall initiate the pattern using the initial primitive and block until a response is received.

3.7.3.8 Requests and Indications

3.7.3.8.1 SUPPORTEDIP

3.7.3.8.1.1 Function

The **SUPPORTEDIP Request** primitive shall be used by the MAL to determine if a Transport supports a specific interaction pattern.

3.7.3.8.1.2 Semantics

SUPPORTEDIP Request shall provide parameters as follows:

(InteractionType)

3.7.3.8.1.3 When Generated

SUPPORTEDIP Request is generated by the MAL when first interacting with a specific Transport.

3.7.3.8.1.4 Effect on Reception

Reception of a **SUPPORTEDIP Request** shall result in a return of a **RESPONSE Indication**.

3.7.3.8.2 RESPONSE

3.7.3.8.2.1 Function

a) The **RESPONSE Indication** primitive shall be used to deliver the response to a **SUPPORTEDIP Request** to the MAL.

b) If the Transport supports the interaction pattern it shall return ‘True’ in the indication, ‘False’ otherwise.

c) It is expected that Transports shall return ‘True’ for interaction patterns SEND, SUBMIT, REQUEST, INVOKE, and PROGRESS, as these do not require any special processing by the Transport layer.

d) Transports shall only return ‘True’ for support of the PUBLISH-SUBSCRIBE interaction pattern if they support the broker aspect natively.

3.7.3.8.2.2 Semantics

RESPONSE Indication shall provide parameters as follows:

(Boolean)

3.7.3.8.2.3 When Generated

RESPONSE Indication shall be generated by the Transport layer in response to a **SUPPORTEDIP Request**.

3.7.3.8.2.4 Effect on Reception

On reception of a **RESPONSE Indication** with the 'False' value for the PUBLISH-SUBSCRIBE pattern, the MAL shall implement the PUBLISH-SUBSCRIBE pattern internally.

3.7.4 TRANSMIT INTERACTION

3.7.4.1 Overview

The TRANSMIT pattern (figure 3-30) is similar to the SUBMIT Interaction Pattern.

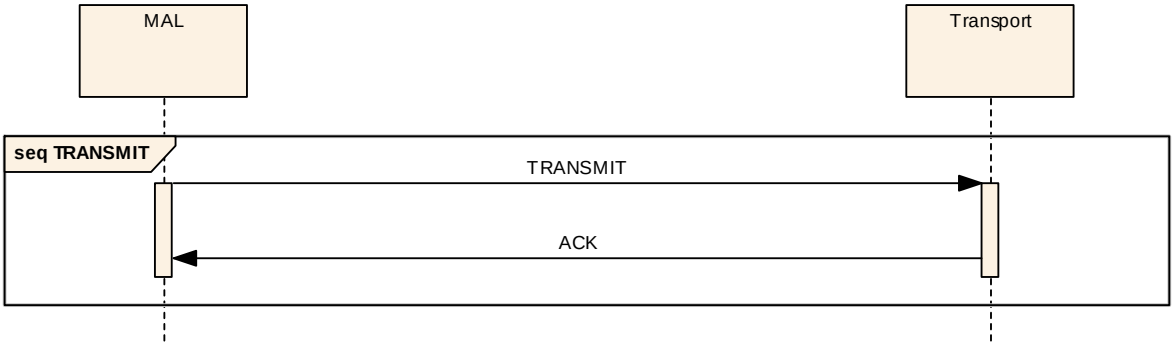

Figure 3-30: TRANSMIT Transport Pattern Message Sequence

3.7.4.2 Description

The TRANSMIT pattern provides a simple transmit/acknowledgement message exchange that is used by a MAL implementation to pass MAL messages to the Transport layer for transmission.

NOTE – It is not expected that this pattern will be implemented via a message, as it is expected to be implemented as a local API used by the MAL to access the Transport layer. It is shown here as a pattern for consistency.

3.7.4.3 Usage

The TRANSMIT pattern is only expected to be used by the MAL for the transmission of MAL Messages. It not expected be used by any other component.

3.7.4.4 Error Handling

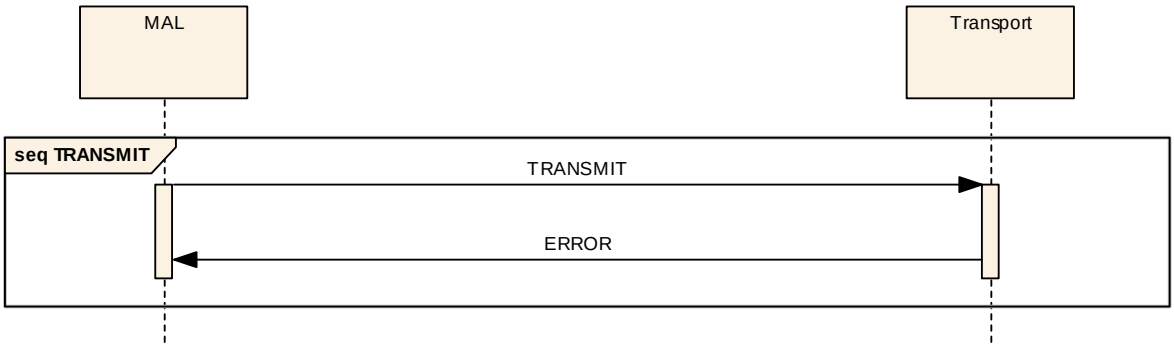

Figure 3-31: TRANSMIT Transport Pattern Error Sequence

a) The acknowledgement shall be replaced with an error message (see 4.4.6) if an error occurs during the processing of the operation (figure 3-31).

b) Either the acknowledgement or the error message shall be returned but never both.

3.7.4.5 Operation Template

The TRANSMIT pattern template is below:

Table 3-51: TRANSMIT Operation Template

Interaction Pattern	TRANSMIT	
Pattern Sequence	Message	Body Type
IN	TRANSMIT	MAL Message
OUT	ACK	

3.7.4.6 Primitives

The pattern is not an abstract interface that provides interoperability. However, it is a function that is required by the MAL.

Table 3-52: TRANSMIT Primitive List

Primitive
TRANSMIT
ACK
ERROR

3.7.4.7 State Charts

No state charts are provided for this pattern. The MAL shall initiate the pattern using the initial primitive and block until a response is received.

3.7.4.8 Requests and Indications

3.7.4.8.1 TRANSMIT

3.7.4.8.1.1 Function

The **TRANSMIT Request** primitive shall be used by the MAL to pass a MAL Message to the Transport layer for transmission.

3.7.4.8.1.2 Semantics

TRANSMIT Request shall provide parameters as follows:

(MAL Message, QoS properties)

3.7.4.8.1.3 When Generated

TRANSMIT Request shall be generated by the MAL when using the Transport layer to transmit messages.

3.7.4.8.1.4 Effect on Reception

a) On reception of a **TRANSMIT Request** a Transport layer shall return an **ACK Indication** if the transmission of the message is successful as far as can be determined initially by the Transport.

b) The determination of success is Transport specific. For example, for a message broker-based transport, this may mean successful transmission to the local broker.

3.7.4.8.2 ACK

3.7.4.8.2.1 Function

The **ACK Indication** primitive shall be used by the Transport layer to indicate to the MAL a successful transmission of a message in response to a **TRANSMIT Request**.

3.7.4.8.2.2 Semantics

ACK Indication does not use any parameters.

3.7.4.8.2.3 When Generated

An **ACK Indication** shall be generated by the Transport layer in response to a **TRANSMIT Request** when the request was successful.

3.7.4.8.2.4 Effect on Reception

The effect on reception of an **ACK Indication** by the MAL is defined in reference [1].

3.7.4.8.3 ERROR

3.7.4.8.3.1 Function

The **ERROR Indication** primitive shall be used by the Transport layer to deliver a transmission failure **ERROR Message** to the MAL.

3.7.4.8.3.2 Semantics

ERROR Indication shall provide parameters as follows:

(ERROR Message Header, StandardError, QoS properties)

3.7.4.8.3.3 When Generated

An **ERROR Indication** shall be generated by the Transport layer if there is an error during transmission.

3.7.4.8.3.4 Effect on Reception

The effect on reception of an **ERROR Indication** by the MAL is defined in reference [1].

3.7.5 TRANSMITMULTIPLE INTERACTION

3.7.5.1 Overview

The TRANSMITMULTIPLE pattern (figure 3-32) is similar to the SUBMIT Interaction Pattern.

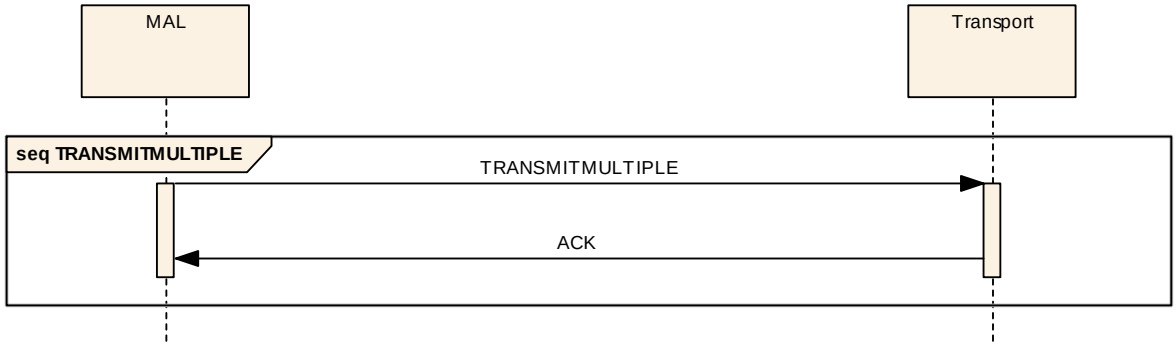

Figure 3-32: TRANSMITMULTIPLE Transport Pattern Message Sequence

3.7.5.2 Description

The TRANSMITMULTIPLE pattern provides a simple transmit/acknowledgement message exchange that is used by a MAL implementation to pass a set of MAL messages to the Transport layer for transmission.

NOTE – It is not expected that this pattern will be implemented via a message, as it is expected to be implemented as a local API used by the MAL to access the Transport layer. It is shown here as a pattern for consistency.

3.7.5.3 Usage

The TRANSMITMULTIPLE pattern is only expected to be used by the MAL for the transmission of MAL Messages. It is not expected to be used by any other component.

3.7.5.4 Error Handling

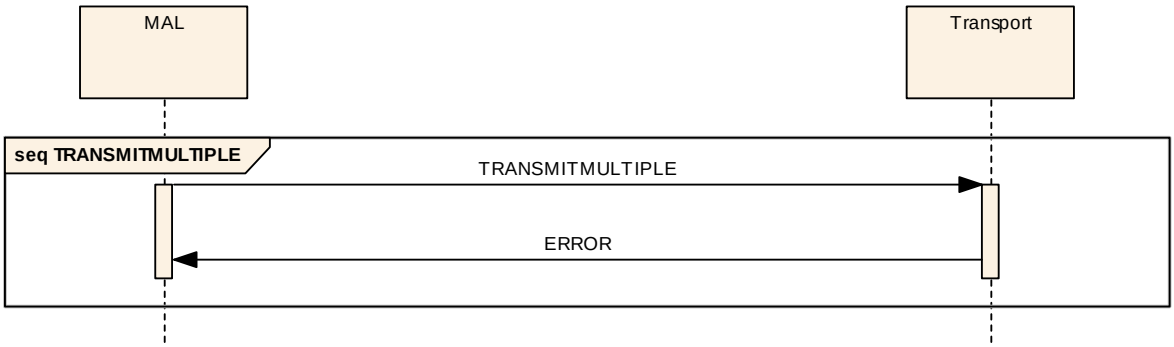

Figure 3-33: TRANSMITMULTIPLE Transport Pattern Error Sequence

a) The acknowledgement shall be replaced with an error message (see 4.4.6) if an error occurs during the processing of the operation (figure 3-33).

b) Either the acknowledgement or the error message shall be returned but never both.

3.7.5.5 Operation Template

The TRANSMITMULTIPLE pattern template is below:

Table 3-53: TRANSMITMULTIPLE Operation Template

Interaction Pattern	TRANSMITMULTIPLE	
Pattern Sequence	Message	Body Type
IN	TRANSMITMULTIPLE	MAL Message List
OUT	ACK	

3.7.5.6 Primitives

The pattern is not an abstract interface that provides interoperability. However, it is a function that is required by the MAL.

Table 3-54: TRANSMITMULTIPLE Primitive List

Primitive
TRANSMITMULTIPLE
ACK
ERROR

3.7.5.7 State Charts

No state charts are provided for this pattern. The MAL shall initiate the pattern using the initial primitive and block until a response is received.

3.7.5.8 Requests and Indications

3.7.5.8.1 TRANSMITMULTIPLE

3.7.5.8.1.1 Function

The **TRANSMITMULTIPLE Request** primitive shall be used by the MAL to pass a list of MAL Messages to the Transport layer for transmission.

3.7.5.8.1.2 Semantics

TRANSMITMULTIPLE Request shall provide parameters as follows:

List of (MAL Message, QoS properties)

3.7.5.8.1.3 When Generated

a) A **TRANSMITMULTIPLE Request** shall be generated by the MAL when using the Transport layer to transmit messages.

b) The pattern is may be used when using a MAL-level broker for a Publish/Subscribe Notify message, i.e., one Notify for each consumer.

3.7.5.8.1.4 Effect on Reception

a) On reception of a **TRANSMITMULTIPLE Request** a Transport layer shall return an **ACK Indication** if the transmission of the messages are all successful as far as can be determined initially.

b) The determination of success shall be Transport specific. For example, for a message broker-based transport, this may mean successful transmission to the local broker.

3.7.5.8.2 ACK

3.7.5.8.2.1 Function

The **ACK Indication** primitive shall be used by the Transport layer to indicate to the MAL a successful transmission of a list of messages in response to a **TRANSMITMULTIPLE Request**.

3.7.5.8.2.2 Semantics

ACK Indication does not use any parameters.

3.7.5.8.2.3 When Generated

An **ACK Indication** shall be generated by the Transport layer in response to a **TRANSMITMULTIPLE Request** when the request was successful.

3.7.5.8.2.4 Effect on Reception

The effect on reception of an **ACK Indication** by the MAL shall be as defined for the TRANSMIT pattern **ACK Indication**.

3.7.5.8.3 ERROR

3.7.5.8.3.1 Function

The **ERROR Indication** primitive shall be used by the Transport layer to deliver a transmission failure **ERROR Message** to the MAL.

3.7.5.8.3.2 Semantics

ERROR Indication shall provide parameters as follows:

List of (ERROR Message Header, StandardError, QoS properties)

3.7.5.8.3.3 When Generated

An **ERROR Indication** shall be generated by the Transport layer if there is an error during transmission of one or more of the messages.

3.7.5.8.3.4 Effect on Reception

a) The effect on reception of an **ERROR Indication** by the MAL shall be as defined for the TRANSMIT pattern **ERROR Indication**.

b) The MAL shall assume a successful transmission for all other messages.

3.7.6 RECEIVE INTERACTION

3.7.6.1 Overview

The RECEIVE pattern is a simple one-way pattern where the Transport layer passes a received message to the MAL (figure 3-34). No acknowledgement is sent by the MAL.

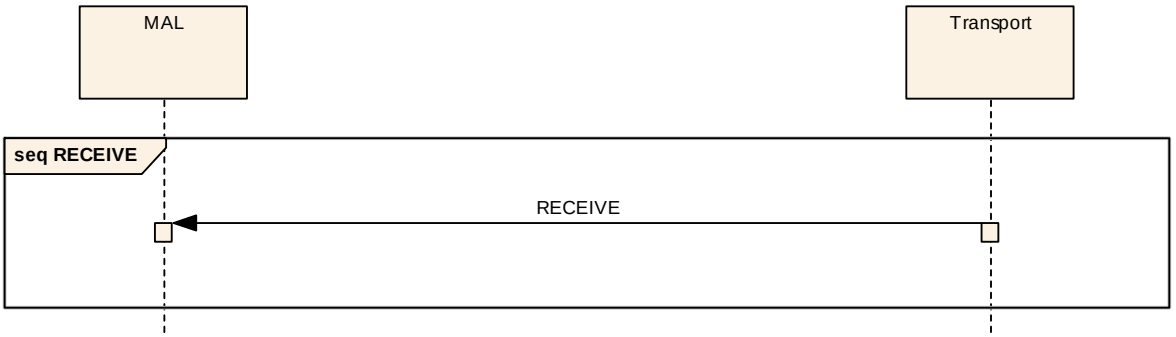

Figure 3-34: RECEIVE Transport Pattern Message Sequence

3.7.6.2 Description

The RECEIVE pattern provides a simple mechanism that is used by the Transport layer to pass a received message to the MAL.

NOTE – It is not expected that this pattern will be implemented via a message, as it is expected to be implemented as a local API used by the MAL to access the Transport layer. It is shown here as a pattern for consistency.

3.7.6.3 Usage

The RECEIVE pattern is only expected to be used by the Transport layer for the passing of message to the MAL. It is not expected to be used by any other component.

3.7.6.4 Error Handling

No Errors shall be raised.

3.7.6.5 Operation Template

The RECEIVE pattern template is below:

Table 3-55: RECEIVE Operation Template

Interaction Pattern	RECEIVE	
Pattern Sequence	Message	Body Type
OUT	RECEIVE	MAL Message

3.7.6.6 Primitives

The pattern is not an abstract interface that provides interoperability. However, it is a function that is required by the MAL.

Table 3-56: RECEIVE Primitive List

Primitive
RECEIVE

3.7.6.7 State Charts

No state charts are provided for this pattern. The Transport layer shall initiate the pattern using an implementation-dependent mechanism, and the MAL is notified via the single Indication.

3.7.6.8 Requests and Indications

3.7.6.8.1 RECEIVE

3.7.6.8.1.1 Function

The **RECEIVE Indication** primitive shall be used by the Transport layer to deliver an incoming message to the MAL.

3.7.6.8.1.2 Semantics

RECEIVE Indication shall provide parameters as follows:

(MAL Message, QoS properties)

3.7.6.8.1.3 When Generated

RECEIVE Indication shall be generated in either one of two situations:

a) upon reception of a message by the Transport layer (this may be an error message);

b) by the Transport layer directly in case of an error in response to a **TRANSMIT Request** or **TRANSMITMULTIPLE Request**. In this case the Transport layer shall create the appropriate MAL Interaction Pattern Error response.

3.7.6.8.1.4 Effect on Reception

The effect on reception of a **RECEIVE Indication** by the MAL is defined in reference [1].

3.7.7 RECEIVEMULTIPLE INTERACTION

3.7.7.1 Overview

The RECEIVEMULTIPLE pattern is a simple one-way pattern where the Transport layer passes a set of received messages to the MAL (figure 3-35). No acknowledgement is sent by the MAL.

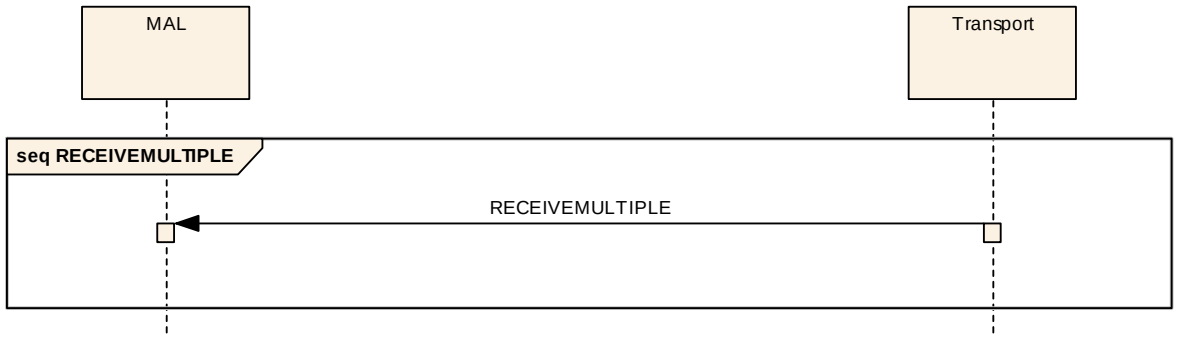

Figure 3-35: RECEIVEMULTIPLE Transport Pattern Message Sequence

3.7.7.2 Description

The RECEIVEMULTIPLE pattern provides a simple mechanism that is used by the Transport layer to pass a set of received messages to the MAL.

NOTE – It is not expected that this pattern will be implemented via a message, as it is expected to be implemented as a local API used by the MAL to access the Transport layer. It is shown here as a pattern for consistency.

3.7.7.3 Usage

The RECEIVEMULTIPLE pattern is only expected to be used by the Transport layer for the passing of messages to the MAL. It is not expected to be used by any other component.

3.7.7.4 Error Handling

No Errors shall be raised.

3.7.7.5 Operation Template

The RECEIVEMULTIPLE pattern template is below:

Table 3-57: RECEIVEMULTIPLE Operation Template

Interaction Pattern	RECEIVEMULTIPLE	
Pattern Sequence	Message	Body Type
OUT	RECEIVEMULTIPLE	MAL Message List

3.7.7.6 Primitives

The pattern is not an abstract interface that provides interoperability. However, it is a function that is required by the MAL.

Table 3-58: RECEIVEMULTIPLE Primitive List

Primitive
RECEIVEMULTIPLE

3.7.7.7 State Charts

No state charts are provided for this pattern. The Transport layer shall initiate the pattern using an implementation-dependent mechanism and the MAL is notified via the single Indication.

3.7.7.8 Requests and Indications

3.7.7.8.1 RECEIVEMULTIPLE

3.7.7.8.1.1 Function

The **RECEIVEMULTIPLE Indication** primitive shall be used by the Transport layer to deliver a list of incoming messages to the MAL.

3.7.7.8.1.2 Semantics

RECEIVEMULTIPLE Indication shall provide parameters as follows:

List of (Header, Message Body, QoS properties)

3.7.7.8.1.3 When Generated

A **RECEIVEMULTIPLE Indication** shall be generated upon reception of a set of messages by the Transport layer (this may include error messages).

3.7.7.8.1.4 Effect on Reception

The **RECEIVEMULTIPLE Indication** is a convenience pattern that allows a Transport layer implementation to pass a set of messages in one interaction rather than in multiple interactions using the **RECEIVE Interaction.** The MAL shall treat each message received as if it had received each one individually.

4 MAL DATA TYPES

4.1 OVERVIEW

4.1.1 GENERAL

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 types and structures defined by the MAL specification and the rules allowed for the combination of these.

4.1.2 FUNDAMENTALS

The base type for all types and structures is Element. Two other fundamental types exist, Composite and Attribute. Only the MAL specification (this document) is allowed to define fundamental types.

Fundament types are represented in a table as illustrated below:

4.1.3 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:

Attribute Name	<<Attribute Name>>
Extends	Attribute
Short form	<<Short Form>>

The types are defined in 4.3, 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.

4.1.4 COMPOSITES

Composites are represented in a table as illustrated below:

Structure Name	TestBody	
Extends	Composite	
Short form	ABC	
Field	Type	Comment
FirstItem	String	Example String item.
SecondItem	Integer	Example Integer item.

Structure Name	ExampleStructure	
Extends	Composite	
Short form	CDE	
Field	Type	Comment
first_item	String	Example String item.
second_item	Integer	Example Integer item.
third_item	TestBody	Contained structure.

The ExampleStructure would decompose to a sequence of:

Field	Type	Comment
first_item	String	from ExampleStructure.
second_item	Integer	from ExampleStructure.
FirstItem	String	from contained TestBody.
SecondItem	Integer	from contained TestBody.

4.1.5 COMPOSITE EXTENSION

A Composite can extend another Composite like below (multiple extension is not permitted nor is extension of a non-abstract composite):

Structure Name	AbstractComposite	
Extends	Composite	
Abstract		
Field	Type	Comment
FirstItem	String	Example String item.
SecondItem	Integer	Example Integer item.

Structure Name	ComplexStructure	
Extends	**AbstractComposite**	
Short form	DEF	
Field	Type	Comment
extra_item	Boolean	Extra Boolean item.
second_item	Integer	Example Integer item.
third_item	TestBody	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:

Field	Type	Comment
FirstItem	String	from AbstractComposite.
SecondItem	Integer	from AbstractComposite.
extra_item	Boolean	from ComplexStructure.
second_item	Integer	from ComplexStructure.
FirstItem	String	from ComplexStructure contained TestBody.
SecondItem	Integer	from ComplexStructure contained TestBody.

It is not permitted to extend a non-abstract composite as an issue arises when the extended composite is used in place of the base composite (which is not marked as abstract). Because there is no indication to the receiving application 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.

4.1.6 CONTAINING ABSTRACT ELEMENTS

It is possible that a composite can contain one of several possible types and that the specific type is not known until the message is created during operations. For example, the type of the value of a telemetered parameter report is not known until the specific parameter is reported. Not allowing a composite to specify that an element is abstract would require a report to be defined for each possible type of parameter.

A composite contains an abstract element by including it just like any normal element; it is the fact that the contained element has been defined as abstract, as shown for type AbstractComposite, that changes the behaviour of the composite. Any composites that extend that abstract composite are possible substitutes in an actual message:

Structure Name	ConcreteCompositeA	
Extends	AbstractComposite	
Short form	CCA	
Field	Type	Comment
ThirdItem	Float	Example Float item.

Structure Name	ConcreteCompositeB	
Extends	AbstractComposite	
Short form	CCB	
Field	Type	Comment
ThirdItem	Time	Example Time item.

Structure Name	OuterComposite	
Extends	Composite	
Short form	OC	
Field	Type	Comment
SomeItem	Time	Example Time item.
AbstractPart	AbstractComposite	This element is the abstract part.

In this example, when an OuterComposite is encoded, the 'AbstractPart' will contain either a ConcreteCompositeA or a ConcreteCompositeB. It is encoding dependent how this is represented, but the encoding must provide enough information for the receiver to decode the correct type. For example, this could be implemented by preceding the contents of the abstract part with the short-form identifier of the actual contained composite.

4.1.7 REPRESENTING ENUMERATIONS

Enumerations are defined sets of possible values. All enumerations are extensions of the fundamental Element and therefore can only be used to replace abstract Elements. They are represented as shown below:

Enumeration Name	ExampleEnum	
Short form	Example only	
Enumeration Value	Short form	Comment
FIRST	1	First enumeration possible value.
SECOND	2	Second enumeration value.
OTHER	3	Etc.
DELETION	4	Object has been deleted.

The short form field is a simplified numerical version of the enumeration. It is expected to be used in efficient encodings and transport mappings.

4.1.8 REPRESENTING LISTS

A list is an arbitrary length sequence of items. All lists are extensions of the basic Composite structure and therefore can be used to replace abstract Composite elements. The definition of a List is as below:

List Name	ExampleList
Short form	Example only
List of	ExampleStructure

The ExampleList is shown as having a 'type' of ExampleStructure and is therefore a list of ExampleStructures. A list has a length part, but whether an actual length field is required is dependent on the protocol. For example, an efficient packet-based protocol may include an initial length field to allow correct decommutation, whereas an XML-based protocol would not require this because XML tags denote the end of the list.

4.1.9 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.

4.2 FUNDAMENTALS

4.2.1 ELEMENT

Element is the base type of all data constructs. All types that make up the common data model are derived from it.

Structure Name	Element
Extends	
Abstract	

4.2.2 ATTRIBUTE

Attribute is the base type of all attributes of the common model. Attributes are contained within Composites and are used to build complex structures that make the data model.

Structure Name	Attribute
Extends	Element
Abstract	

4.2.3 COMPOSITE

Composite is the base structure for composite structures that contain a set of elements.

Structure Name	Composite
Extends	Element
Abstract	

4.3 ATTRIBUTES

4.3.1 BLOB

The Blob structure is used to store binary object attributes. It is a variable-length, unbounded, octet array. 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.

Attribute Name	Blob
Extends	Attribute
Short form	L

4.3.2 BOOLEAN

The Boolean structure is used to store Boolean attributes. Possible values are ‘True’ or ‘False’.

Attribute Name	Boolean
Extends	Attribute
Short form	B

4.3.3 DURATION

The Duration structure is used to store Duration attributes. It represents a length of time in seconds. It may contain a fractional component.

Attribute Name	Duration
Extends	Attribute
Short form	D

4.3.4 FLOAT

The Float structure is used to store floating point attributes using the IEEE 754 32-bit range.

Three special values exist for this type: POSITIVE_INFINITY, NEGATIVE_INFINITY, and NaN (Not A Number).

Attribute Name	Float
Extends	Attribute
Short form	F

4.3.5 DOUBLE

The Double structure is used to store floating point attributes using the IEEE 754 64-bit range.

Three special values exist for this type: POSITIVE_INFINITY, NEGATIVE_INFINITY, and NaN (Not A Number).

Attribute Name	Double
Extends	Attribute
Short form	Z

4.3.6 IDENTIFIER

The Identifier structure is used to store an identifier and can be used for indexing. It is a variable-length, unbounded, UTF16 string.

Attribute Name	Identifier
Extends	Attribute
Short form	I

4.3.7 OCTET

The Octet structure is used to store eight-bit signed attributes. The permitted range is -128 to 127.

Attribute Name	Octet
Extends	Attribute
Short form	A

4.3.8 SHORT

The Short structure is used to store 16-bit signed attributes. The permitted range is -32768 to 32767.

Attribute Name	Short
Extends	Attribute
Short form	H

4.3.9 INTEGER

The Integer structure is used to store 32-bit signed attributes. The permitted range is -2147483648 to 2147483647.

Attribute Name	Integer
Extends	Attribute
Short form	N

4.3.10 LONG

The Long structure is used to store 64-bit signed attributes. The permitted range is -9223372036854775808 to 9223372036854775807.

Attribute Name	Long
Extends	Attribute
Short form	Y

4.3.11 STRING

The String structure is used to store String attributes. It is a variable-length, unbounded, UTF-16 Unicode string.

Attribute Name	String
Extends	Attribute
Short form	S

4.3.12 TIME

The Time structure is used to store absolute time attributes. It represents an absolute date and time to millisecond resolution.

Attribute Name	Time
Extends	Attribute
Short form	T

4.3.13 FINETIME

The FineTime structure is used to store high-resolution absolute time attributes. It represents an absolute date and time to picosecond resolution.

Attribute Name	FineTime
Extends	Attribute
Short form	C

4.3.14 URI

The URI structure is used to store URI addresses. It is a variable-length, unbounded, UTF16 string.

Attribute Name	URI
Extends	Attribute
Short form	U

4.4 DATA STRUCTURES

4.4.1 INTERACTIONTYPE ENUMERATION

InteractionType is an enumeration holding the possible interaction pattern types.

Enumeration Name	InteractionType	
Short form	G	
Enumeration Value	Short form	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.

4.4.2 SESSIONTYPE ENUMERATION

SessionType is an enumeration holding the session types.

Enumeration Name	SessionType	
Short form	g	
Enumeration Value	Short form	Comment
LIVE	1	Used for Live sessions.
SIMULATION	2	Used for Simulation sessions.
REPLAY	3	Used for Replay Sessions.

4.4.3 QOSLEVEL ENUMERATION

QoSLevel is an enumeration holding the possible QoS levels.

Enumeration Name	QoSLevel	
Short form	O	
Enumeration Value	Short form	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.

4.4.4 UPDATETYPE ENUMERATION

UpdateType is an enumeration holding the possible Update types.

Enumeration Name	UpdateType	
Short form	M	
Enumeration Value	Short form	Comment
CREATION	1	Update is notification of the creation of the item.
UPDATE	2	Update is just a periodic update of the item and has not changed its value.
MODIFICATION	3	Update is for a changed value or modification of the item.
DELETION	4	Update is notification of the removal of the item.

4.4.5 MESSAGEHEADER

The MessageHeader structure is used to hold all fields that are passed for each message exchanged between a consumer and provider.

Structure Name	MessageHeader	
Extends	Composite	
Short form	x	
Field	Type	Comment
URIfrom	URI	Message Source URI.
authenticationId	Blob	Source Authentication Identifier.
URIto	URI	Message Destination URI.
timestamp	Time	Message generation timestamp.
QoSlevel	QoSLevel	The QoS level of the message.
priority	Integer	The QoS priority of the message.
domain	DomainIdentifier	Domain of the message.
networkZone	Identifier	Network zone of the message.
session	SessionType	Type of session of the message.
sessionName	Identifier	Name of the session of the message. Shall be ‘LIVE’ if session type is LIVE.
interactionType	InteractionType	Interaction Pattern Type.
interactionStage	Octet	Interaction Pattern Stage.
transactionId	Identifier	Unique to consumer.
area	Identifier	Service Area Identifier.
service	Identifier	Service Identifier.
operation	Identifier	Service Operation Identifier.
version	Octet	Service version.
isError	Boolean	‘True’ if this is an error message; else ‘False’.

4.4.6 STANDARDERROR

This basic StandardError structure allows an operation to return an error code. The service specification shall define which error codes may be returned for a specific operation and also what extra information structure is provided.

If no extra information is provided by an error, then the extraInformation field should be set to NULL.

Structure Name	StandardError	
Extends	Composite	
Short form	X	
Field	Type	Comment
errorNumber	Integer	Operation-specific error code.
extraInformation	Element	Allows provision of extra error-specific values if required. Normally left empty.

4.4.7 DOMAINIDENTIFIER

A DomainIdentifier is a list of identifiers that is defined as a separate type from the normal IdentifierList so that encodings and transports can handle it in a specialised mechanism dependent on their architecture.

List Name	DomainIdentifier
Short form	W
List of	Identifier

The most significant domain part is listed first in the list (for example Agency) and each subsequent domain identifier in the list narrows the preceding domain.

Each Identifier part of the Domain is allowed the full range of Identifier values with the restriction that it is NOT allowed to contain the '.' character. A DomainIdentifier can also be represented using a single Identifier. Each part of the DomainIdentifier is concatenated using the '.' character with the most significant first. For example:

Agency.Mission.Craft.Subsystem

4.4.8 SUBSCRIPTION

The Subscription structure is used when subscribing for updates using the PUBSUB interaction pattern. It contains a single identifier that identifies the subscription being defined and a set of entities being requested.

Structure Name	Subscription	
Extends	Composite	
Short form	R	
Field	Type	Comment
subscriptionId	Identifier	The identifier of this subscription.
entities	EntityRequestList	The list of entities that are being subscribed for by this identified subscription.

4.4.9 ENTITYREQUEST

The EntityRequest structure is used when subscribing for updates using the PUBSUB interaction pattern.

Structure Name	EntityRequest	
Extends	Composite	
Short form	Q	
Field	Type	Comment
subDomain	DomainIdentifier	Optional subdomain identifier that is appended to the Message Header domain identifier when requesting entities in a subdomain of this domain. May be NULL.
allAreas	Boolean	If set to True, then all updates regardless of Area shall be sent.
allServices	Boolean	If set to True, then all updates regardless of Service shall be sent.

allOperations	Boolean	If set to True, then all updates regardless of Operation shall be sent.
entityKeys	EntityKeyList	The list of entities to be monitored.
onlyOnChange	Boolean	The Boolean denotes that only change updates are to be sent rather than all updates.

4.4.10 ENTITYKEY

The EntityKey structure is used to identify an entity in the PUBSUB interaction pattern.

Structure Name	EntityKey	
Extends	Composite	
Short form	K	
Field	Type	Comment
firstSubKey	Identifier	The first sub-key of the key.
secondSubKey	Identifier	The second sub-key of the key.
thirdSubKey	Identifier	The third sub-key of the key.
fourthSubKey	Identifier	The fourth sub-key of the key.

4.4.11 SUBSCRIPTIONUPDATE

The SubscriptionUpdate structure is used for updates sent to a consumer of a PUBSUB interaction pattern.

Structure Name	SubscriptionUpdate	
Extends	Composite	
Short form	r	
Field	Type	Comment
subscriptionId	Identifier	The identifier of this subscription.

updateList	UpdateList	The list of object updates that are being updated by this identified subscription.

4.4.12 UPDATE

The Update structure is used by updates using the PUBSUB interaction pattern. It must be extended by the specific service to be used.

Structure Name	Update	
Extends	Composite	
Abstract		
Field	Type	Comment
timestamp	Time	Creation timestamp of the update.
sourceURI	URI	URI of the source of the update, usually a PUBSUB provider.
updateType	UpdateType	Type of update being reported.
key	EntityKey	The key of the entity; shall not contain the wildcard character of '*'.

4.4.13 IDBOOLEANPAIR

IdBooleanPair is a simple pair type of an identifier and Boolean value.

Structure Name	IdBooleanPair	
Extends	Composite	
Short form	V	
Field	Type	Comment
Id	Identifier	The Identifier value.
value	Boolean	The Boolean value.

4.4.14 PAIR

Pair is a simple composite structure for holding pairs. The pairs can be user-defined attributes.

Structure Name	Pair	
Extends	Composite	
Short form	P	
Field	Type	Comment
first	Attribute	The attribute value for the first element of this pair.
second	Attribute	The attribute value for the second element of this pair.

4.4.15 NAMEDVALUE

The NamedValue structure represents a simple pair type of an identifier and abstract attribute value.

Structure Name	NamedValue	
Extends	Composite	
Short form	J	
Field	Type	Comment
name	Identifier	The Identifier value.
value	Attribute	The Attribute value.

4.4.16 ENTITYREQUESTLIST

List Name	EntityRequestList
Short form	q
List of	EntityRequest

4.4.17 SUBSCRIPTIONUPDATELIST

List Name	SubscriptionUpdateList
Short form	w
List of	SubscriptionUpdate

4.4.18 UPDATELIST

List Name	UpdateList
Short form	m
List of	Update

4.4.19 ENTITYKEYLIST

List Name	EntityKeyList
Short form	k
List of	EntityKey

4.4.20 STRINGLIST

List Name	StringList
Short form	s
List of	String

4.4.21 BLOBLIST

List Name	BlobList
Short form	l
List of	Blob

4.4.22 BOOLEANLIST

List Name	BooleanList
Short form	b
List of	Boolean

4.4.23 DURATIONLIST

List Name	DurationList
Short form	d
List of	Duration

4.4.24 FLOATLIST

List Name	FloatList
Short form	f
List of	Float

4.4.25 DOUBLELIST

List Name	DoubleList
Short form	z
List of	Double

4.4.26 IDENTIFIERLIST

List Name	IdentifierList
Short form	i
List of	Identifier

4.4.27 OCTETLIST

List Name	OctetList
Short form	a
List of	Octet

4.4.28 SHORTLIST

List Name	ShortList
Short form	h
List of	Short

4.4.29 INTEGERLIST

List Name	IntegerList
Short form	n
List of	Integer

4.4.30 LONGLIST

List Name	LongList
Short form	y
List of	Long

4.4.31 TIMELIST

List Name	TimeList
Short form	t
List of	Time

4.4.32 FINETIMELIST

List Name	FineTimeList
Short form	c
List of	FineTime

4.4.33 URILIST

List Name	URIList
Short form	u
List of	URI

4.4.34 QOSLEVELLIST

List Name	QoSLevelList
Short form	o
List of	QoSLevel

4.4.35 PAIRLIST

List Name	PairList
Short form	p
List of	Pair

4.4.36 IDBOOLEANLIST

List Name	IdBooleanList
Short form	v
List of	IdBooleanPair

4.4.37 NAMEDVALUELIST

List Name	NamedValueList
Short form	j
List of	NamedValue

5 MAL ERRORS

Each operation shall list any errors, specific to that operation over and above any standard errors, that can be raised by an implementation. Error codes for an operation should start at zero '0' and increment; standard errors start at 65536, 0x10000 in hex, and increment; operation-specific errors shall therefore remain inside the inclusive range of 0 to 65535.

The following table lists the standard errors:

Table 5-1: Standard MAL Error Codes

Error	Error #	Comments
DELIVERY_FAILED	65536	Confirmed communication error.
DELIVERY_TIMEDOUT	65537	Unconfirmed communication error.
DELIVERY_DELAYED	65538	Message queued somewhere awaiting contact.
DESTINATION_UNKNOWN	65539	Destination cannot be contacted.
DESTINATION_TRANSIENT	65540	Destination middleware reports destination application does not exist.
DESTINATION_LOST	65541	Destination lost halfway through conversation.
AUTHENTICATION_FAIL	65542	A failure to authenticate the message correctly.
AUTHORISATION_FAIL	65543	A failure in the MAL to authorise the message.
ENCRYPTION_FAIL	65544	A failure in the MAL to encrypt/decrypt the message.
UNSUPPORTED_AREA	65545	The destination does not support the service area.
UNSUPPORTED_OPERATION	65546	The destination does not support the operation.
UNSUPPORTED_VERSION	65547	The destination does not support the service version.

Error	Error #	Comments
BAD_ENCODING	65548	The destination was unable to decode the message.
INTERNAL	65549	An internal error has occurred.
UNKNOWN	65550	Operation specific.
INCORRECT_STATE	65551	The destination was not in the correct state for the received message.
TOO_MANY	65552	Maximum number of subscriptions or providers of a broker has been exceeded.
SHUTDOWN	65553	The component is being shutdown.

Only two errors are possible from the MAL and below with regards to Authentication and Authorisation. They are concerned with message-level issues and do not cover login issues such as an incorrect username/password combination.

NOTE – The authentication and authorisation failure messages are very generic in nature; it is possible that a weakness in a specific protocol or authentication mechanism could be exploited if too much information is returned about the specific nature of an authentication or authorisation failure.

6 SERVICE SPECIFICATION XML

6.1 GENERAL

The following section defines a normative XML schema for MO service specification. The use of XML for service specification provides a machine readable format rather than the text based document format.

Subsection 6.3 defines the XML Schema that is used to validate the actual XML service specifications. Subsection 6.4 provides the normative XML for the MAL specification; it uses the XML schema from the previous section.

6.2 SCHEMA RULES

Only the MAL specification is permitted to define data types of ‘fundamental’ and ‘attribute’.

6.3 SERVICE XML SCHEMA

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.ccsds.org/schema/ServiceSchema"
        xmlns:smc="http://www.ccsds.org/schema/ServiceSchema"
        elementFormDefault="qualified">

 <xsd:element name="specification" type="smc:SpecificationType">
  <xsd:annotation>
   <xsd:documentation xml:lang="en">Root node of the document.</xsd:documentation>
  </xsd:annotation>
  <xsd:unique name="areaNameCheck">
   <xsd:annotation>
    <xsd:documentation xml:lang="en">Ensures that Area names are unique.</xsd:documentation>
   </xsd:annotation>
   <xsd:selector xpath="smc:area"/>
   <xsd:field    xpath="@name"/>
  </xsd:unique>
  <xsd:unique name="areaNumberCheck">
   <xsd:annotation>
    <xsd:documentation xml:lang="en">Ensures that Area numbers are unique.</xsd:documentation>
   </xsd:annotation>
   <xsd:selector xpath="smc:area"/>
   <xsd:field    xpath="@number"/>
  </xsd:unique>
  <xsd:unique name="typeName">
   <xsd:annotation>
    <xsd:documentation xml:lang="en">Ensures that Type names are unique.</xsd:documentation>
   </xsd:annotation>
   <xsd:selector xpath=".//smc:fundamental|.//smc:composite|.//smc:list|.//smc:attribute|.//smc:enumeration"/>
   <xsd:field    xpath="@name"/>
  </xsd:unique>
  <xsd:unique name="typeShortFormCheck">
   <xsd:annotation>
    <xsd:documentation xml:lang="en">Ensures that type short forms are unique.</xsd:documentation>
   </xsd:annotation>
   <xsd:selector xpath=".//smc:composite|.//smc:list|.//smc:baseType|.//smc:enumeration"/>
   <xsd:field    xpath="@shortForm"/>
  </xsd:unique>
```

```
  <xsd:unique name="errorName">
    <xsd:annotation>
      <xsd:documentation xml:lang="en">Ensures that error names are unique.</xsd:documentation>
    </xsd:annotation>
    <xsd:selector xpath=".//smc:error"/>
    <xsd:field    xpath="@name"/>
  </xsd:unique>
  <xsd:unique name="errorNumberCheck">
    <xsd:annotation>
      <xsd:documentation xml:lang="en">Ensures that Error numbers are unique.</xsd:documentation>
    </xsd:annotation>
    <xsd:selector xpath=".//smc:error"/>
    <xsd:field    xpath="@number"/>
  </xsd:unique>
 </xsd:element>

 <xsd:complexType name="SpecificationType">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">Top level element that contains one or more Area specifications.</xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="area" type="smc:AreaType" maxOccurs="unbounded">
      <xsd:unique name="serviceName">
        <xsd:annotation>
          <xsd:documentation xml:lang="en">Ensures that Service names are unique to an Area.</xsd:documentation>
        </xsd:annotation>
        <xsd:selector xpath="smc:service"/>
        <xsd:field    xpath="@name"/>
      </xsd:unique>
      <xsd:unique name="serviceNumber">
        <xsd:annotation>
          <xsd:documentation xml:lang="en">Ensures that Service numbers are unique to an Area.</xsd:documentation>
        </xsd:annotation>
        <xsd:selector xpath="smc:service"/>
        <xsd:field    xpath="@number"/>
      </xsd:unique>
    </xsd:element>
  </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="AreaType">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">Element that represents an Area, contains one or more Service specifications and optionally data
type and error definitions.</xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="service" type="smc:ServiceType" minOccurs="0" maxOccurs="unbounded">
      <xsd:unique name="capabilityNumber">
        <xsd:annotation>
          <xsd:documentation xml:lang="en">Capability numbers inside a service should be unique.</xsd:documentation>
        </xsd:annotation>
        <xsd:selector xpath="smc:capabilitySet"/>
        <xsd:field    xpath="@number"/>
      </xsd:unique>
      <xsd:unique name="operationName">
        <xsd:annotation>
          <xsd:documentation xml:lang="en">Operation names inside a service should be unique.</xsd:documentation>
        </xsd:annotation>
        <xsd:selector xpath="smc:capabilitySet/*"/>
        <xsd:field    xpath="@name"/>
      </xsd:unique>
      <xsd:unique name="operationNumber">
        <xsd:annotation>
          <xsd:documentation xml:lang="en">Operation numbers inside a service should be unique.</xsd:documentation>
```

```
        </xsd:annotation>
        <xsd:selector xpath="smc:capabilitySet/*"/>
        <xsd:field    xpath="@number"/>
       </xsd:unique>
     </xsd:element>
     <xsd:element name="dataTypes" type="smc:AreaDataTypeList" maxOccurs="1" minOccurs="0"/>
     <xsd:element name="errors" type="smc:ErrorDefinitionList" minOccurs="0"/>
   </xsd:sequence>
   <xsd:attribute name="name" type="smc:NameString" use="required"/>
   <xsd:attribute name="number" type="xsd:nonNegativeInteger" use="required"/>
  </xsd:complexType>

  <xsd:complexType name="ServiceType">
   <xsd:annotation>
     <xsd:documentation xml:lang="en">Element represents a Service specification inside an Area. May also optionally include data type
and error definitions.</xsd:documentation>
   </xsd:annotation>
   <xsd:sequence>
     <xsd:element name="capabilitySet" type="smc:CapabilitySetType" maxOccurs="unbounded" minOccurs="0"/>
     <xsd:element name="dataTypes" type="smc:DataTypeList" maxOccurs="1" minOccurs="0"/>
     <xsd:element name="errors" type="smc:ErrorDefinitionList" minOccurs="0"/>
   </xsd:sequence>
   <xsd:attribute name="name" type="smc:NameString" use="required"/>
   <xsd:attribute name="number" type="xsd:nonNegativeInteger" use="required"/>
   <xsd:attribute name="version" type="xsd:positiveInteger" use="required"/>
  </xsd:complexType>

  <xsd:complexType name="CapabilitySetType">
   <xsd:annotation>
     <xsd:documentation xml:lang="en">Element that represents a Capability set, contains one or more operation
specification.</xsd:documentation>
   </xsd:annotation>
   <xsd:sequence>
     <xsd:choice maxOccurs="unbounded">
       <xsd:element name="sendIP" type="smc:SendOperationType"/>
       <xsd:element name="submitIP" type="smc:SubmitOperationType"/>
       <xsd:element name="requestIP" type="smc:RequestOperationType"/>
       <xsd:element name="invokeIP" type="smc:InvokeOperationType"/>
       <xsd:element name="progressIP" type="smc:ProgressOperationType"/>
       <xsd:element name="pubsubIP" type="smc:PubSubOperationType"/>
     </xsd:choice>
   </xsd:sequence>
   <xsd:attribute name="number" type="xsd:nonNegativeInteger" use="required"/>
  </xsd:complexType>

  <xsd:complexType name="SendOperationType">
   <xsd:annotation>
     <xsd:documentation xml:lang="en">Element represents a SEND operation.</xsd:documentation>
   </xsd:annotation>
   <xsd:complexContent>
     <xsd:extension base="smc:OperationType">
       <xsd:sequence>
         <xsd:element name="messages">
           <xsd:complexType>
             <xsd:sequence>
               <xsd:element name="send" type="smc:OptionalElementReferenceType"/>
             </xsd:sequence>
           </xsd:complexType>
         </xsd:element>
       </xsd:sequence>
     </xsd:extension>
   </xsd:complexContent>
  </xsd:complexType>
```

```
<xsd:complexType name="SubmitOperationType">
 <xsd:annotation>
  <xsd:documentation xml:lang="en">Element represents a SUBMIT operation.</xsd:documentation>
 </xsd:annotation>
 <xsd:complexContent>
  <xsd:extension base="smc:OperationType">
   <xsd:sequence>
    <xsd:element name="messages">
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element name="submit" type="smc:OptionalElementReferenceType"/>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
    <xsd:element name="errors" type="smc:OperationErrorList" minOccurs="0"></xsd:element>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="RequestOperationType">
 <xsd:annotation>
  <xsd:documentation xml:lang="en">Element represents a REQUEST operation.</xsd:documentation>
 </xsd:annotation>
 <xsd:complexContent>
  <xsd:extension base="smc:OperationType">
   <xsd:sequence>
    <xsd:element name="messages">
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element name="request" type="smc:OptionalElementReferenceType"/>
       <xsd:element name="response" type="smc:OptionalElementReferenceType"/>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
    <xsd:element name="errors" type="smc:OperationErrorList" minOccurs="0"></xsd:element>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="InvokeOperationType">
 <xsd:annotation>
  <xsd:documentation xml:lang="en">Element represents a INVOKE operation.</xsd:documentation>
 </xsd:annotation>
 <xsd:complexContent>
  <xsd:extension base="smc:OperationType">
   <xsd:sequence>
    <xsd:element name="messages">
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element name="invoke" type="smc:OptionalElementReferenceType"/>
       <xsd:element name="acknowledgement" type="smc:OptionalElementReferenceType"/>
       <xsd:element name="response" type="smc:OptionalElementReferenceType"/>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
    <xsd:element name="errors" type="smc:OperationErrorList" minOccurs="0"></xsd:element>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="ProgressOperationType">
```

```
  <xsd:annotation>
    <xsd:documentation xml:lang="en">Element represents a PROGRESS operation.</xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="smc:OperationType">
      <xsd:sequence>
        <xsd:element name="messages">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="progress" type="smc:OptionalElementReferenceType"/>
              <xsd:element name="acknowledgement" type="smc:OptionalElementReferenceType"/>
              <xsd:element name="update" type="smc:OptionalElementReferenceType"/>
              <xsd:element name="response" type="smc:OptionalElementReferenceType"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="errors" type="smc:OperationErrorList" minOccurs="0"></xsd:element>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:complexType name="PubSubOperationType">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">Element represents a PUBSUB operation.</xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="smc:OperationType">
      <xsd:sequence>
        <xsd:element name="messages">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="publishNotify" type="smc:OptionalElementReferenceType"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="errors" type="smc:OperationErrorList" minOccurs="0"></xsd:element>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:complexType name="OperationType" abstract="true">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">Base type that the operations extend, defines the common aspects of all
operations.</xsd:documentation>
  </xsd:annotation>
  <xsd:attribute name="name" type="smc:NameString" use="required"/>
  <xsd:attribute name="number" type="xsd:nonNegativeInteger" use="required"/>
  <xsd:attribute name="supportInReplay" type="xsd:boolean" use="required"/>
  <xsd:attribute name="comment" type="xsd:string" use="optional"/>
 </xsd:complexType>

 <xsd:complexType name="AreaDataTypeList">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">List that contains one or more data type definitions including those only allowed in the MAL
Area.</xsd:documentation>
  </xsd:annotation>
  <xsd:choice minOccurs="1" maxOccurs="unbounded">
    <xsd:element name="fundamental" type="smc:FundamentalType">
      <xsd:annotation>
        <xsd:documentation xml:lang="en">Only to be used for Element, Attribute, and Composite. Shall only be present in the MAL Area
type definitions.</xsd:documentation>
      </xsd:annotation>
```

```
    </xsd:element>
    <xsd:element name="attribute" type="smc:AttributeType">
     <xsd:annotation>
      <xsd:documentation xml:lang="en">Shall only be present in the MAL Area type definitions.</xsd:documentation>
     </xsd:annotation>
    </xsd:element>
    <xsd:element name="composite" type="smc:CompositeType">
     <xsd:unique name="areaFieldName">
      <xsd:annotation>
       <xsd:documentation xml:lang="en">Field names inside a Composite must be unique.</xsd:documentation>
      </xsd:annotation>
      <xsd:selector xpath="smc:field"/>
      <xsd:field    xpath="@name"/>
     </xsd:unique>
    </xsd:element>
    <xsd:element name="enumeration" type="smc:EnumerationType">
     <xsd:unique name="areaEnumerationItemNumber">
      <xsd:annotation>
       <xsd:documentation xml:lang="en">Item shortForm numbers inside an Enumeration must be unique.</xsd:documentation>
      </xsd:annotation>
      <xsd:selector xpath="smc:item"/>
      <xsd:field    xpath="@shortForm"/>
     </xsd:unique>
    </xsd:element>
    <xsd:element name="list" type="smc:ListType"/>
   </xsd:choice>
  </xsd:complexType>

  <xsd:complexType name="DataTypeList">
   <xsd:annotation>
    <xsd:documentation xml:lang="en">List that contains one or more data type definitions.</xsd:documentation>
   </xsd:annotation>
   <xsd:choice minOccurs="1" maxOccurs="unbounded">
    <xsd:element name="composite" type="smc:CompositeType">
     <xsd:unique name="fieldName">
      <xsd:annotation>
       <xsd:documentation xml:lang="en">Field names inside a Composite must be unique.</xsd:documentation>
      </xsd:annotation>
      <xsd:selector xpath="smc:field"/>
      <xsd:field    xpath="@name"/>
     </xsd:unique>
    </xsd:element>
    <xsd:element name="enumeration" type="smc:EnumerationType">
     <xsd:unique name="enumerationItemNumber">
      <xsd:annotation>
       <xsd:documentation xml:lang="en">Item shortForm numbers inside an Enumeration must be unique.</xsd:documentation>
      </xsd:annotation>
      <xsd:selector xpath="smc:item"/>
      <xsd:field    xpath="@shortForm"/>
     </xsd:unique>
    </xsd:element>
    <xsd:element name="list" type="smc:ListType"/>
   </xsd:choice>
  </xsd:complexType>

  <xsd:complexType name="OperationErrorList">
   <xsd:annotation>
    <xsd:documentation xml:lang="en">Represents that list of Errors that an operation may raise, these may reference existing error
definitions or define new ones.</xsd:documentation>
   </xsd:annotation>
   <xsd:choice maxOccurs="unbounded">
    <xsd:element name="error" type="smc:ErrorDefinitionType"/>
    <xsd:element name="errorRef" type="smc:ErrorReferenceType"/>
   </xsd:choice>
```

```
 </xsd:complexType>

 <xsd:complexType name="ErrorDefinitionList">
  <xsd:annotation>
   <xsd:documentation xml:lang="en">Represents a list of Error definitions.</xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
   <xsd:element name="error" type="smc:ErrorDefinitionType" maxOccurs="unbounded"/>
  </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="ErrorDefinitionType">
  <xsd:annotation>
   <xsd:documentation xml:lang="en">Used to define a new Error condition</xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
   <xsd:element name="extraInformation" type="smc:ElementReferenceWithCommentType" minOccurs="0" maxOccurs="1">
    <xsd:annotation>
     <xsd:documentation xml:lang="en">Optional element that is used to define the type of extra information that is returned with an
error.</xsd:documentation>
    </xsd:annotation>
   </xsd:element>
  </xsd:sequence>
  <xsd:attribute name="name" type="smc:NameString" use="required"/>
  <xsd:attribute name="number" type="xsd:nonNegativeInteger" use="required"/>
  <xsd:attribute name="comment" type="xsd:string" use="optional"/>
 </xsd:complexType>

 <xsd:complexType name="ErrorReferenceType">
  <xsd:annotation>
   <xsd:documentation xml:lang="en">Used to reference an existing Error condition</xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
   <xsd:extension base="smc:ElementReferenceWithCommentType">
    <xsd:sequence>
     <xsd:element name="extraInformation" type="smc:ElementReferenceWithCommentType" minOccurs="0" maxOccurs="1">
      <xsd:annotation>
       <xsd:documentation xml:lang="en">Optional element that is used to define the type of extra information that is returned with an
error, replaces any type defined in the referenced error.</xsd:documentation>
      </xsd:annotation>
     </xsd:element>
    </xsd:sequence>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:complexType name="FundamentalType">
  <xsd:annotation>
   <xsd:documentation xml:lang="en">Represents one of the three fundamental types, element, attribute, or
composite.</xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
   <xsd:element name="extends" type="smc:ElementReferenceType" minOccurs="0" maxOccurs="1">
    <xsd:annotation>
     <xsd:documentation xml:lang="en">use to show the relationship between attribute/composite and element.</xsd:documentation>
    </xsd:annotation>
   </xsd:element>
  </xsd:sequence>
  <xsd:attribute name="name" type="smc:NameString" use="required"/>
  <xsd:attribute name="comment" type="xsd:string" use="optional"/>
 </xsd:complexType>

 <xsd:complexType name="AttributeType">
  <xsd:annotation>
```

```
    <xsd:documentation xml:lang="en">Defines a new attribute type. Shall only be present in the MAL Area.</xsd:documentation>
   </xsd:annotation>
   <xsd:attribute name="name" type="smc:NameString" use="required"/>
   <xsd:attribute name="shortForm" type="smc:NameString" use="required"/>
   <xsd:attribute name="comment" type="xsd:string" use="optional"/>
  </xsd:complexType>

  <xsd:complexType name="CompositeType">
   <xsd:annotation>
    <xsd:documentation xml:lang="en">Represents a composite type.</xsd:documentation>
   </xsd:annotation>
   <xsd:sequence>
    <xsd:element name="extends" type="smc:ElementReferenceType" minOccurs="0" maxOccurs="1">
     <xsd:annotation>
      <xsd:documentation xml:lang="en">Which Composite this type extends, if this is not present then the base Composite is
implied.</xsd:documentation>
     </xsd:annotation>
    </xsd:element>
    <xsd:element name="field" type="smc:NamedElementReferenceWithCommentType" minOccurs="0" maxOccurs="unbounded"/>
   </xsd:sequence>
   <xsd:attribute name="name" type="smc:NameString" use="required"/>
   <xsd:attribute name="shortForm" type="smc:NameString" use="optional">
    <xsd:annotation>
     <xsd:documentation xml:lang="en">If no shortForm is provided then this is an Abstract composite type.</xsd:documentation>
    </xsd:annotation>
   </xsd:attribute>
   <xsd:attribute name="comment" type="xsd:string" use="optional"/>
  </xsd:complexType>

  <xsd:complexType name="EnumerationType">
   <xsd:annotation>
    <xsd:documentation xml:lang="en">Represents an Enumeration type. By definition an Enumeration extends Element and therefore this
aspect cannot be set.</xsd:documentation>
   </xsd:annotation>
   <xsd:sequence>
    <xsd:element name="item" minOccurs="1" maxOccurs="unbounded">
     <xsd:complexType>
      <xsd:attribute name="value" type="smc:NameString" use="required"/>
      <xsd:attribute name="shortForm" type="xsd:nonNegativeInteger" use="required"/>
      <xsd:attribute name="comment" type="xsd:string" use="optional"/>
     </xsd:complexType>
    </xsd:element>
   </xsd:sequence>
   <xsd:attribute name="name" type="smc:NameString" use="required"/>
   <xsd:attribute name="shortForm" type="smc:NameString" use="required"/>
   <xsd:attribute name="comment" type="xsd:string" use="optional"/>
  </xsd:complexType>

  <xsd:complexType name="ListType">
   <xsd:annotation>
    <xsd:documentation xml:lang="en">Represents a List type. By definition a List extends Composite and therefore this aspect cannot be
set.</xsd:documentation>
   </xsd:annotation>
   <xsd:complexContent>
    <xsd:extension base="smc:NamedElementReferenceWithCommentType">
     <xsd:attribute name="shortForm" type="smc:NameString" use="required"/>
    </xsd:extension>
   </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="NamedElementReferenceWithCommentType">
   <xsd:complexContent>
    <xsd:extension base="smc:ElementReferenceWithCommentType">
     <xsd:attribute name="name" type="smc:NameString" use="required"/>
```

```
      <xsd:attribute name="canBeNull" type="xsd:boolean" default="true"/>
    </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:complexType name="ElementReferenceWithCommentType">
  <xsd:complexContent>
   <xsd:extension base="smc:ElementReferenceType">
    <xsd:attribute name="comment" type="xsd:string" use="optional"/>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:complexType name="OptionalElementReferenceType">
  <xsd:sequence>
   <xsd:element name="type" type="smc:ElementSMCReferenceType" minOccurs="0"/>
  </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="ElementReferenceType">
  <xsd:sequence>
   <xsd:element name="type" type="smc:ElementSMCReferenceType"/>
  </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="ElementSMCReferenceType">
  <xsd:annotation>
   <xsd:documentation xml:lang="en">Allows another element in the specification to be referenced, if the area or the service attributes are
missing then the area and/or service of the refering element shall be assumed.</xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
   <xsd:extension base="smc:NamedElementType">
    <xsd:attribute name="area" type="smc:NameString" use="required"/>
    <xsd:attribute name="service" type="smc:NameString" use="optional"/>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:complexType name="NamedElementType">
  <xsd:attribute name="name" type="smc:NameString" use="required"/>
 </xsd:complexType>

 <xsd:simpleType name="NameString">
  <xsd:restriction base="xsd:string">
   <xsd:minLength value="1"/>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:schema>
```

6.4 MAL SERVICE XML

```
<?xml version="1.0" encoding="UTF-8"?>
<smc:specification xmlns:smc="http://www.ccsds.org/schema/ServiceSchema"
                   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <smc:area name="MAL" number="0">
  <smc:dataTypes>
   <smc:fundamental name="Attribute">
    <smc:extends>
     <smc:type name="Element" area="MAL"/>
    </smc:extends>
   </smc:fundamental>
   <smc:fundamental name="Composite">
    <smc:extends>
     <smc:type name="Element" area="MAL"/>
```

```
      </smc:extends>
    </smc:fundamental>
    <smc:fundamental name="Element"/>
    <smc:attribute name="Blob" shortForm="L"/>
    <smc:attribute name="Boolean" shortForm="B"/>
    <smc:attribute name="Double" shortForm="Z"/>
    <smc:attribute name="Duration" shortForm="D"/>
    <smc:attribute name="FineTime" shortForm="C"/>
    <smc:attribute name="Float" shortForm="F"/>
    <smc:attribute name="Identifier" shortForm="I"/>
    <smc:attribute name="Integer" shortForm="N"/>
    <smc:attribute name="Long" shortForm="Y"/>
    <smc:attribute name="Octet" shortForm="A"/>
    <smc:attribute name="Short" shortForm="H"/>
    <smc:attribute name="String" shortForm="S"/>
    <smc:attribute name="Time" shortForm="T"/>
    <smc:attribute name="URI" shortForm="U"/>
    <smc:enumeration name="InteractionType" shortForm="G" comment="Enumeration definition holding the possible interaction pattern
types.">
      <smc:item value="SEND" shortForm="1" comment="Used for Send interactions"/>
      <smc:item value="SUBMIT" shortForm="2" comment="Used for Submit interactions"/>
      <smc:item value="REQUEST" shortForm="3" comment="Used for Request interactions"/>
      <smc:item value="INVOKE" shortForm="4" comment="Used for Invoke interactions"/>
      <smc:item value="PROGRESS" shortForm="5" comment="Used for Progress interactions"/>
      <smc:item value="PUBSUB" shortForm="6" comment="Used for Publish/Subscribe interactions"/>
    </smc:enumeration>
    <smc:enumeration name="QoSLevel" shortForm="O" comment="Enumeration definition holding the possible QoS levels.">
      <smc:item value="BESTEFFORT" shortForm="1" comment="Used for Best Effort QoS Level"/>
      <smc:item value="ASSURED" shortForm="2" comment="Used for Assured QoS Level"/>
      <smc:item value="QUEUED" shortForm="3" comment="Used for Queued QoS Level"/>
      <smc:item value="TIMELY" shortForm="4" comment="Used for Timely QoS Level"/>
    </smc:enumeration>
    <smc:enumeration name="SessionType" shortForm="g" comment="Enumeration definition holding the session types.">
      <smc:item value="LIVE" shortForm="1" comment="Used for Live sessions"/>
      <smc:item value="SIMULATION" shortForm="2" comment="Used for Simulation sessions"/>
      <smc:item value="REPLAY" shortForm="3" comment="Used for Replay sessions"/>
    </smc:enumeration>
    <smc:enumeration name="UpdateType" shortForm="M" comment="Enumeration definition holding the possible Update types.">
      <smc:item value="CREATION" shortForm="1" comment="Update is notification of the creation of the item."/>
      <smc:item value="UPDATE" shortForm="2" comment="Update is just a periodic update of the item and has not changed its value."/>
      <smc:item value="MODIFICATION" shortForm="3" comment="Update is for a changed value or modification of the item."/>
      <smc:item value="DELETION" shortForm="4" comment="Update is notification of the removal of the item."/>
    </smc:enumeration>
    <smc:list name="BlobList" shortForm="l">
      <smc:type name="Blob" area="MAL"/>
    </smc:list>
    <smc:list name="BooleanList" shortForm="b">
      <smc:type name="Boolean" area="MAL"/>
    </smc:list>
    <smc:list name="DomainIdentifier" shortForm="W">
      <smc:type name="Identifier" area="MAL"/>
    </smc:list>
    <smc:list name="DoubleList" shortForm="z">
      <smc:type name="Double" area="MAL"/>
    </smc:list>
    <smc:list name="DurationList" shortForm="d">
      <smc:type name="Duration" area="MAL"/>
    </smc:list>
    <smc:list name="EntityKeyList" shortForm="k">
      <smc:type name="EntityKey" area="MAL"/>
    </smc:list>
    <smc:list name="EntityRequestList" shortForm="q">
      <smc:type name="EntityRequest" area="MAL"/>
    </smc:list>
```

```
    <smc:list name="FineTimeList" shortForm="c">
      <smc:type name="FineTime" area="MAL"/>
    </smc:list>
    <smc:list name="FloatList" shortForm="f">
      <smc:type name="Float" area="MAL"/>
    </smc:list>
    <smc:list name="IdBooleanList" shortForm="v">
      <smc:type name="IdBooleanPair" area="MAL"/>
    </smc:list>
    <smc:list name="IdentifierList" shortForm="i">
      <smc:type name="Identifier" area="MAL"/>
    </smc:list>
    <smc:list name="IntegerList" shortForm="n">
      <smc:type name="Integer" area="MAL"/>
    </smc:list>
    <smc:list name="LongList" shortForm="y">
      <smc:type name="Long" area="MAL"/>
    </smc:list>
    <smc:list name="NamedValueList" shortForm="j">
      <smc:type name="NamedValue" area="MAL"/>
    </smc:list>
    <smc:list name="OctetList" shortForm="a">
      <smc:type name="Octet" area="MAL"/>
    </smc:list>
    <smc:list name="PairList" shortForm="p">
      <smc:type name="Pair" area="MAL"/>
    </smc:list>
    <smc:list name="QoSLevelList" shortForm="o">
      <smc:type name="QoSLevel" area="MAL"/>
    </smc:list>
    <smc:list name="ShortList" shortForm="h">
      <smc:type name="Short" area="MAL"/>
    </smc:list>
    <smc:list name="StringList" shortForm="s">
      <smc:type name="String" area="MAL"/>
    </smc:list>
    <smc:list name="SubscriptionUpdateList" shortForm="w">
      <smc:type name="SubscriptionUpdate" area="MAL"/>
    </smc:list>
    <smc:list name="TimeList" shortForm="t">
      <smc:type name="Time" area="MAL"/>
    </smc:list>
    <smc:list name="UpdateList" shortForm="m">
      <smc:type name="Update" area="MAL"/>
    </smc:list>
    <smc:list name="URIList" shortForm="u">
      <smc:type name="URI" area="MAL"/>
    </smc:list>
    <smc:composite name="EntityKey" shortForm="K" comment="The EntityKey structure is used to identify an entity in the PUBSUB
interaction pattern.">
      <smc:extends>
        <smc:type name="Composite" area="MAL"/>
      </smc:extends>
      <smc:field name="firstSubKey" comment="The first sub-key of the key.">
        <smc:type name="Identifier" area="MAL"/>
      </smc:field>
      <smc:field name="secondSubKey" comment="The second sub-key of the key.">
        <smc:type name="Identifier" area="MAL"/>
      </smc:field>
      <smc:field name="thirdSubKey" comment="The third sub-key of the key.">
        <smc:type name="Identifier" area="MAL"/>
      </smc:field>
      <smc:field name="fourthSubKey" comment="The fourth sub-key of the key.">
        <smc:type name="Identifier" area="MAL"/>
```

```
      </smc:field>
    </smc:composite>
    <smc:composite name="EntityRequest" shortForm="Q" comment="This structure is used when subscribing for updates using the PUBSUB interaction pattern.">
      <smc:extends>
        <smc:type name="Composite" area="MAL"/>
      </smc:extends>
      <smc:field name="subDomain" canBeNull="true" comment=" Optional sub domain identifier that is appended to the Message Header domain identifier when requesting entities in a sub domain of this domain.">
        <smc:type name="DomainIdentifier" area="MAL"/>
      </smc:field>
      <smc:field name="allAreas" canBeNull="false" comment="If set to True, then all updates regardless of Area shall be sent.">
        <smc:type name="Boolean" area="MAL"/>
      </smc:field>
      <smc:field name="allServices" canBeNull="false" comment="If set to True, then all updates regardless of Service shall be sent.">
        <smc:type name="Boolean" area="MAL"/>
      </smc:field>
      <smc:field name="allOperations" canBeNull="false" comment="If set to True, then all updates regardless of Operation shall be sent.">
        <smc:type name="Boolean" area="MAL"/>
      </smc:field>
      <smc:field name="entityKeys" canBeNull="false" comment="The list of entities to be monitored.">
        <smc:type name="EntityKeyList" area="MAL"/>
      </smc:field>
      <smc:field name="onlyOnChange" canBeNull="false" comment="The Boolean denotes that only change updates to be sent rather than all updates.">
        <smc:type name="Boolean" area="MAL"/>
      </smc:field>
    </smc:composite>
    <smc:composite name="IdBooleanPair" shortForm="V" comment="Simple pair type of an identifier and Boolean value.">
      <smc:extends>
        <smc:type name="Composite" area="MAL"/>
      </smc:extends>
      <smc:field name="id" comment="The Identifier value.">
        <smc:type name="Identifier" area="MAL"/>
      </smc:field>
      <smc:field name="value" comment="The Boolean value.">
        <smc:type name="Boolean" area="MAL"/>
      </smc:field>
    </smc:composite>
    <smc:composite name="MessageHeader" shortForm="x" comment="The MessageHeader structure is used to hold all fields that are passed for each message exchanged between a consumer and provider. See 3.4 for more information.">
      <smc:extends>
        <smc:type name="Composite" area="MAL"/>
      </smc:extends>
      <smc:field name="URIfrom" comment="Message Source URI">
        <smc:type name="URI" area="MAL"/>
      </smc:field>
      <smc:field name="authenticationId" comment="Source Authentication Credentials">
        <smc:type name="Blob" area="MAL"/>
      </smc:field>
      <smc:field name="URIto" canBeNull="false" comment="Message Destination URI">
        <smc:type name="URI" area="MAL"/>
      </smc:field>
      <smc:field name="timestamp" canBeNull="false" comment="Message generation timestamp">
        <smc:type name="Time" area="MAL"/>
      </smc:field>
      <smc:field name="QoSlevel" canBeNull="false" comment="The QoS level of the message">
        <smc:type name="QoSLevel" area="MAL"/>
      </smc:field>
      <smc:field name="priority" canBeNull="false" comment="The QoS priority of the message">
        <smc:type name="Integer" area="MAL"/>
      </smc:field>
      <smc:field name="domain" canBeNull="false" comment="Domain of the message">
        <smc:type name="DomainIdentifier" area="MAL"/>
```

```
      </smc:field>
      <smc:field name="networkZone" canBeNull="false" comment="Network zone of the message">
        <smc:type name="Identifier" area="MAL"/>
      </smc:field>
      <smc:field name="session" canBeNull="false" comment="Type of session of the message">
        <smc:type name="SessionType" area="MAL"/>
      </smc:field>
      <smc:field name="sessionName" canBeNull="false" comment="Name of the session of the message. Shall be 'LIVE' if session type is
LIVE.">
        <smc:type name="Identifier" area="MAL"/>
      </smc:field>
      <smc:field name="interactionType" canBeNull="false" comment="Interaction Pattern Type">
        <smc:type name="InteractionType" area="MAL"/>
      </smc:field>
      <smc:field name="interactionStage" canBeNull="false" comment="Interaction Pattern Stage">
        <smc:type name="Octet" area="MAL"/>
      </smc:field>
      <smc:field name="transactionId" comment="Unique to consumer">
        <smc:type name="Identifier" area="MAL"/>
      </smc:field>
      <smc:field name="area" canBeNull="false" comment="Service Area Identifier">
        <smc:type name="Identifier" area="MAL"/>
      </smc:field>
      <smc:field name="service" canBeNull="false" comment="Service Identifier">
        <smc:type name="Identifier" area="MAL"/>
      </smc:field>
      <smc:field name="operation" canBeNull="false" comment="Service Operation Identifier">
        <smc:type name="Identifier" area="MAL"/>
      </smc:field>
      <smc:field name="version" canBeNull="false" comment="Service version">
        <smc:type name="Octet" area="MAL"/>
      </smc:field>
      <smc:field name="isError" canBeNull="false" comment="True if this is an error message else False.">
        <smc:type name="Boolean" area="MAL"/>
      </smc:field>
    </smc:composite>
    <smc:composite name="NamedValue" shortForm="J" comment="This structure represents a simple pair type of an identifier and
abstract attribute value which could be a new user defined type.">
      <smc:extends>
        <smc:type name="Composite" area="MAL"/>
      </smc:extends>
      <smc:field name="name" canBeNull="false" comment="The Identifier value.">
        <smc:type name="Identifier" area="MAL"/>
      </smc:field>
      <smc:field name="value" comment="The Attribute value.">
        <smc:type name="Attribute" area="MAL"/>
      </smc:field>
    </smc:composite>
    <smc:composite name="Pair" shortForm="P" comment="Simple composite structure for holding pairs. The pairs can be user defined
attributes.">
      <smc:extends>
        <smc:type name="Composite" area="MAL"/>
      </smc:extends>
      <smc:field name="first" comment="The attribute value for the first element of this pair.">
        <smc:type name="Attribute" area="MAL"/>
      </smc:field>
      <smc:field name="second" comment="The attribute value for the second element of this pair.">
        <smc:type name="Attribute" area="MAL"/>
      </smc:field>
    </smc:composite>
    <smc:composite name="StandardError" shortForm="X" comment="This basic structure allows an operation to return an error code. The
service specification shall define which error codes may be returned for a specific operation and also what extra information structure is
provided. If no extra information is provided by an error then the extraInformation field should be set to NULL.">
      <smc:extends>
```

```
        <smc:type name="Composite" area="MAL"/>
       </smc:extends>
       <smc:field name="errorNumber" canBeNull="false" comment="Operation-specific error code">
        <smc:type name="Integer" area="MAL"/>
       </smc:field>
       <smc:field name="extraInformation" comment="Allows provision of extra error-specific values if required. Normally left empty.">
        <smc:type name="Element" area="MAL"/>
       </smc:field>
     </smc:composite>
     <smc:composite name="Subscription" shortForm="R" comment="This structure is used when subscribing for updates using the
PUBSUB interaction pattern. It contains a single identifier that identifies the subscription being defined and a set of entities being
requested.">
       <smc:extends>
        <smc:type name="Composite" area="MAL"/>
       </smc:extends>
       <smc:field name="subscriptionId" canBeNull="false" comment="The identifier of this subscription">
        <smc:type name="Identifier" area="MAL"/>
       </smc:field>
       <smc:field name="entities" canBeNull="false" comment="The list of entities that are being subscribed for by this identified
subscription.">
        <smc:type name="EntityRequestList" area="MAL"/>
       </smc:field>
     </smc:composite>
     <smc:composite name="SubscriptionUpdate" shortForm="r" comment="This structure is used for updates sent to a consumer of a
PUBSUB interaction pattern.">
       <smc:extends>
        <smc:type name="Composite" area="MAL"/>
       </smc:extends>
       <smc:field name="subscriptionId" canBeNull="false" comment="The identifier of this subscription">
        <smc:type name="Identifier" area="MAL"/>
       </smc:field>
       <smc:field name="updateList" canBeNull="false" comment="The list of object updates that are being updated by this identified
subscription.">
        <smc:type name="UpdateList" area="MAL"/>
       </smc:field>
     </smc:composite>
     <smc:composite name="Update" comment="This structure is used by updates using the PUBSUB interaction pattern. It must be
extended by the specific service to be used.">
       <smc:extends>
        <smc:type name="Composite" area="MAL"/>
       </smc:extends>
       <smc:field name="timestamp" canBeNull="false" comment="Creation timestamp of the update.">
        <smc:type name="Time" area="MAL"/>
       </smc:field>
       <smc:field name="sourceURI" canBeNull="false" comment="URI of the source of the update, usually a PUBSUB provider.">
        <smc:type name="URI" area="MAL"/>
       </smc:field>
       <smc:field name="updateType" canBeNull="false" comment="Type of update being reported.">
        <smc:type name="UpdateType" area="MAL"/>
       </smc:field>
       <smc:field name="key" canBeNull="false" comment="The key of the entity.">
        <smc:type name="EntityKey" area="MAL"/>
       </smc:field>
     </smc:composite>
   </smc:dataTypes>
   <smc:errors>
     <smc:error name="DELIVERY_FAILED" number="65536" comment="Confirmed communication error"/>
     <smc:error name="DELIVERY_TIMEDOUT" number="65537" comment="Unconfirmed communication error"/>
     <smc:error name="DELIVERY_DELAYED" number="65538" comment="Message queued somewhere awaiting contact"/>
     <smc:error name="DESTINATION_UNKNOWN" number="65539" comment="Destination cannot be contacted"/>
     <smc:error name="DESTINATION_TRANSIENT" number="65540" comment="Destination middleware reports destination application
does not exist"/>
     <smc:error name="DESTINATION_LOST" number="65541" comment="Destination lost halfway through conversation"/>
     <smc:error name="AUTHENTICATION_FAIL" number="65542" comment="A failure to authenticate the message correctly"/>
```

```
    <smc:error name="AUTHORISATION_FAIL" number="65543" comment="A failure in the MAL to authorise the message"/>
    <smc:error name="ENCRYPTION_FAIL" number="65544" comment="A failure in the MAL to encrypt/decrypt the message"/>
    <smc:error name="UNSUPPORTED_AREA" number="65545" comment="The destination does not support the service area"/>
    <smc:error name="UNSUPPORTED_OPERATION" number="65546" comment="The destination does not support the operation"/>
    <smc:error name="UNSUPPORTED_VERSION" number="65547" comment="The destination does not support the service version"/>
    <smc:error name="BAD_ENCODING" number="65548" comment="The destination was unable to decode the message"/>
    <smc:error name="INTERNAL" number="65549" comment="An internal error has occurred"/>
    <smc:error name="UNKNOWN" number="65550" comment="Operation specific"/>
    <smc:error name="INCORRECT_STATE" number="65551" comment="The destination was not in the correct state for the received
message."/>
    <smc:error name="TOO_MANY" number="65552" comment="Maximum number of subscriptions or providers of a broker has been
exceeded"/>
    <smc:error name="SHUTDOWN" number="65553" comment="The component is being shutdown"/>
   </smc:errors>
  </smc:area>
</smc:specification>
```

7 CONFORMANCE MATRIX

This section provides the Conformance Matrix for implementations of the MAL. A MAL will be considered to be 'conformant' if the mandatory elements identified in the matrix are implemented as described in this Recommended Standard.

Table 7-1: Conformance Matrix

Interaction Pattern	Optional/Mandatory
SEND	Mandatory
SUBMIT	Mandatory
REQUEST	Mandatory
INVOKE	Mandatory
PROGRESS	Mandatory
PUBSUB	Mandatory

ANNEX A

DEFINITION OF ACRONYMS

(INFORMATIVE)

AMS	CCSDS Asynchronous Message Service
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BLOB	Binary Large Object
CCSDS	Consultative Committee for Space Data Statndards
IEEE	Institute of Electrical and Electronics Engineers
LAN	Local Area Network
MAL	Message Abstract Layer
MCS	Mission Control System
MO	Mission Operations
NaN	Not A Number
QoS	Quality of Service
RPC	Remote Procedure Call
SM&C	CCSDS Spacecraft Monitor & Control
URI	Universal Resource Identifier
UTF	Unicode Transformation Format
XML	eXtensible Markup Language

ANNEX B

INFORMATIVE REFERENCES

(INFORMATIVE)

[B1] *Mission Operations Services Concept.* Report Concerning Space Data System Standards, CCSDS 520.0-G-3. Green Book. Issue 3. Washington, D.C.: CCSDS, [forthcoming].

NOTE – Normative references are listed in 1.6.