MAL prototyping plan

This document describes the plan to implement the MAL prototype which is specified in the sections 2.1 and 5 of the document untitled “CCSDS SMC Document roadmap V2-3”.

The first part presents how the reference test bed (see section 5.2 of the roadmap) is implemented on top of the Java Virtual machine.

The second part gives an overview of the test cases describing the main test objectives.

The third part specifies the test services to be used and how the provider side shall implement them.

The fourth part specifies the test scenarios, cases and procedures.

The fifth part specifies the test data structure used by both sides (consumer and provider).

The sixth part sums up the requirements established by the MAL book and indicates which test procedure verifies each requirement.

The seventh part gives the test result output format.

1 Reference test bed over Java

This section explains how each layer of the reference test bed is instantiated.

1.1 API

Both sides of the test bed, “implementation #1” and “implementation #2” share the same Java MAL API.

1.2 Test service (Test Application)

This is the top layer of the test bed as shown in the roadmap. Those services are defined once with the standard XML format. Stubs are generated from this definition towards the target API.

1.3 Transport

The transport module is shared by both MAL implementations.

However two transport modules are to be used:

1. MAL/Joram
2. MAL/AMQP

1.4 Test framework

It is proposed to use a test framework based on JUnit and Ant.

The coordination of the tests is explained in section 4.

2 Tests overview

The tests aim at checking that two implementations of the MAL can interoperate as specified by the MAL book. The same transport layer is used by both implementations.

The following topics shall be checked:

- Interaction patterns
- Data structures
- Standard errors
- Quality of Service

2.1 Interaction patterns

The tests check that the two MAL implementations interoperate as follows:

- They manage to communicate with each other through each interaction pattern.
- They correctly assign and interpret the header values of a MAL message.
- They are able to interact with the broker provided by the other implementation.

2.1.1 Communication

Each interaction pattern shall be initiated by a consumer using one MAL implementation and handled by a provider using the other MAL implementation.

Some scenarios are to be defined in order to go through all possible transitions as specified by the IP sequence diagrams and state charts, i.e. each interaction stage shall be tested. If an expected transition is missing then the test fails.

2.1.2 Message header

The header values are checked on the receiver side. The values are checked as follows:

- Some headers shall have the same value as on the sender side, e.g. domain identifier, network zone, URIs, QoS level.
- Some headers shall be assigned with a particular value defined in the MAL book.

Moreover some constraints taken from the MAL book are checked. For example the session name shall be “LIVE” if the session type is LIVE

Remarks:

- The interaction type and stage headers are implicitly checked by the communication test (see

2.1.1).

- All the enumerated values shall be tested, e.g. the QoS levels Best Effort, Assured, Queued and Timely.

2.1.3 Broker

The creation of a subscription is tested for a given service, operation, domain, network zone, session and entity key request. For example, the test checks:

- The interpretation done by the broker of the entity request expression (*, NULL).
- Errors (e.g. unknown entity key)
- The subscription to the same entity several times. The updates shall be notified only once per entity.
- That a subscription can be overridden by another (with the same name).
- That the Transaction Id must be taken from the initial Register message regardless of subsequent Register messages that have been sent by the consumer to modify an active subscription. If all subscriptions are deregistered, and therefore the active subscription has ended, and then a Register message is sent, and therefore a new active subscription is started, it shall be this newer Transaction Id that shall then be used for subsequent Notify messages.
- That an error can be published.

Both types of broker are tested: private and shared.

2.2 Data structures

The data structure encoding is tested through an IP, for example Send: the consumer encodes the structure and the provider decodes it.

All the structures defined in the MAL book shall be tested. Abstract structures are tested through a sub type (either one already specified or specifically defined for the test)

The test checks that the decoded structure is the same as the encoded one.

Some more verifications have to be done. Here are some examples:

- Polymorphism and NULL value shall be tested.
- For enumerations, the test should check each enumerated values.
- The ordering of the list elements shall be preserved.

2.3 Errors

Some errors may or may not be raised by the transport layer. So it is not possible to ensure at the MAL

level that such errors are to be raised in some conditions. It depends on the transport layer, e.g. the “delivery delayed” error may not be raised by a RPC based transport.

As a consequence a fake transport module is to be implemented in order to raise all the MAL errors according to some particular conditions.

A test security module also has to be implemented in order to raise the security errors.

3 MALPrototype Service Specification

3.1 IPTest Service

This service aims at testing each Interaction Pattern (IP). It provides one operation for Send, Submit, Request, Invoke and Progress. The input parameter is an IPTestDefinition that contains:

- The parameters used by the consumer in order to initiate the interaction. These parameters enable the provider to check whether the received message header is correct or not.
- A list of interaction transitions expected by the consumer.

An operation 'getResult' is provided in order to enable the consumer to get:

- the interaction transaction identifier
- and the assertions evaluated on the provider side during an interaction.

Finally four operations 'monitor', 'addPublishedEntities', 'publishUpdates' and 'publishError' are provided in order to test the Pub/Sub interaction.

3.1.1 Check message header

When the provider receives a MAL message (initiating an interaction) it has to check that the header is the same as the expected header.

The expected header is deduced from the IPTestDefinition as follows:

Field	Assigned with
URIfrom	Field 'consumerURI' of the IPTestDefinition.
authenticationId	Field 'authenticationId' of the IPTestDefinition.
URIto	The provider's URI.
timestamp	Field 'timestamp' of the IPTestDefinition.
QoSlevel	Field 'qos' of the IPTestDefinition.
priority	Field 'priority' of the IPTestDefinition.
domain	Field 'domain' of the IPTestDefinition.
networkZone	Field 'networkZone' of the IPTestDefinition.
session	Field 'session' of the IPTestDefinition.
sessionName	Field 'sessionName' of the IPTestDefinition.
interactionType	Interaction type used by the operation, e.g. the operation “send” uses a SEND interaction type.
interactionStage	1
transactionId	Not assigned

area	The test area name “MALPrototype”.
service	The test service name “IPTest”.
operation	The operation name.
version	The test service version.
isError	False

The provider creates an InteractionKey and checks that it is unique by adding it into a hash table. The InteractionKey table is never cleaned during the life time of the provider.

For each field, except 'timestamp' and 'transactionId', the following assertion is made:

Assertion	
Info	Result
Check header field '<field name>'	True if the value of the received header field is equal to the expected value fields False otherwise.

For 'timestamp' and 'transactionId', the assertions are:

Assertion	
Info	Result
Check header field 'timestamp'	True if the value of the received header timestamp is greater than the expected header timestamp. False otherwise.
Check header field 'transactionId'	True if the InteractionKey is unique. False otherwise.

3.1.2 Check transitions

The provider has to trigger the transitions that are expected by the consumer. Those transitions are specified by the attribute 'transitions' of the operation parameter IPTestDefinition.

For each IPTestTransition, the provider calls the primitive that triggers the transition, catches potential errors and evaluate the following assertion:

Assertion	
Info	Result
Check transition <transition type>	True if: • No error is raised and no error is expected. • An error is raised and its code is equal to the

	expected error code. False otherwise.

3.1.3 Check Publish header

The Publish header is obtained through the test transport module (see section 4.5).

The expected header is built from the parameter TestPublication as follows:

Field	Assigned with
URIfrom	The provider's URI
authenticationId	Field 'authenticationId' of the TestPublication
URIto	The broker's URI
timestamp	Current time before the publication
QoSlevel	Field 'qos' of the TestPublication
priority	Field 'priority' of the TestPublication
domain	Field 'domain' of the TestPublication
networkZone	Field 'networkZone' of the TestPublication
session	Field 'session' of the TestPublication
sessionName	Field 'sessionName' of the TestPublication
interactionType	Pub/Sub
interactionStage	3
transactionId	Not assigned.
area	The test area name “MALPrototype”.
service	The test service name “IPTest”.
operation	The operation name.
version	The test service version.
isError	False

For each field, except 'timestamp' and 'transactionId', the following assertion is made:

Assertion	
Info	Result
Check header field '<field name>'	True if the value of the received header field is equal to the expected value fields False otherwise.

For 'timestamp' the assertion is:

Assertion	
Info	Result
Check header field 'timestamp'	True if the value of the received header timestamp is greater than the expected header timestamp. False otherwise.

The field 'transactionId' is not used by the Publish request so it is not checked

3.1.4 Check Publish Error header

The Publish Error header is obtained through the test transport module (see section 4.5).

The expected header is built from the parameter TestPublication in the same way as in section 3.1.3 except for the field:

Field	Assigned with
isError	True

The same assertions as in section 3.1.3 are made.

3.1.5 General

The service interface is described below:

Area Identifier	Service Identifier	Area Number	Service Number	Service Version
MALPrototype	IPTest	<test area nb>	0	1
Interaction Pattern	**Operation Name**	**Operation Number**	**Support in replay**	**Capability Set**
SEND	send	100	No	100
SUBMIT	submit	101	No	
REQUEST	request	102	No	
INVOKE	invoke	103	No	
PROGRESS	progress	104	No	
PUBSUB	monitor	105	No	
REQUEST	getResult	106	No	101
SEND	addPublishedEntity	107	No	102

SEND	publishUpdates	108	No	103
SEND	publishError	109	No	

3.1.6 OPERATION: send

3.1.6.1 General

This operation cleans the assertions table and check that the header of the received message is the same as the one expected (see 3.1.1).

Operation Name	send	
Interaction Pattern	SEND	
IP Sequence	Message	Field Type
IN	Send	IPTestDefinition

3.1.7 OPERATION: submit

3.1.7.1 General

This operation cleans the assertions table and checks that the header of the received message is the same as the one expected (see 3.1.1). Moreover it triggers the transitions specified by the IPTestDefinition (see 3.1.2).

Operation Name	submit	
Interaction Pattern	SUBMIT	
IP Sequence	Message	Field Type
IN	Submit	IPTestDefinition

3.1.7.2 Errors

The following error can be raised by this operation:

Error	Error #	Comments
TEST_ERROR	70000	Fake error for testing.

3.1.8 OPERATION: request

3.1.8.1 General

This operation cleans the assertions table and checks that the header of the received message is the same

as the one expected (see 3.1.1). Moreover it triggers the transitions specified by the IPTestDefinition (see 3.1.2).

Operation Name	request	
Interaction Pattern	REQUEST	
IP Sequence	Message	Field Type
IN	Request	IPTestDefinition
OUT	Response	MAL::String

3.1.8.2 Errors

The following error can be raised by this operation:

Error	Error #	Comments
TEST_ERROR	70000	Fake error for testing.

3.1.9 OPERATION: invoke

3.1.9.1 General

This operation cleans the assertions table and checks that the header of the received message is the same as the one expected (see 3.1.1). Moreover it triggers the transitions specified by the IPTestDefinition (see 3.1.2).

Operation Name	invoke	
Interaction Pattern	INVOKE	
IP Sequence	Message	Field Type
IN	Request	IPTestDefinition
OUT	Acknowledgement	MAL::String
OUT	Response	MAL::String

3.1.9.2 Errors

The following error can be raised by this operation:

Error	Error #	Comments
TEST_ERROR	70000	Fake error for testing.

3.1.10 OPERATION: progress

3.1.10.1 General

This operation cleans the assertions table and checks that the header of the received message is the same as the one expected (see 3.1.1). Moreover it triggers the transitions specified by the IPTestDefinition (see 3.1.2).

Operation Name	progress	
Interaction Pattern	PROGRESS	
IP Sequence	Message	Field Type
IN	Request	IPTestDefinition
OUT	Acknowledgement	MAL::String
OUT	Update	MAL::Integer
OUT	Response	MAL::String

3.1.10.2 Errors

The following error can be raised by this operation:

Error	Error #	Comments
TEST_ERROR	<TEST ERROR CODE>	Fake error for testing.

3.1.11 OPERATION: getResult

3.1.11.1 General

This operation returns an IPTestResult.

Operation Name	getResult	
Interaction Pattern	REQUEST	
IP Sequence	Message	Field Type
IN	Request	MAL::Element
OUT	Response	IPTestResult

3.1.12 OPERATION: monitor

3.1.12.1 General

This operation initiates a Pub/Sub interaction. It is not implemented by the service provider but by a broker.

Operation Name	monitor	
Interaction Pattern	PUBLISH-SUBSCRIBE	
IP Sequence	Message	Field Type
OUT	Publish/Notify	TestUpdate

3.1.12.2 Errors

The following error can be raised by this operation:

Error	Error #	Comments
UNKNOWN		One or more of the entities identified in the registration do not exist.

3.1.13 OPERATION: addPublishedEntities

3.1.13.1 General

This operation makes the provider declare to publish the specified entities.

Operation Name	addPublishedEntities	
Interaction Pattern	SEND	
IP Sequence	Message	Field Type
IN	Send	MAL::EntityKeyList

3.1.14 OPERATION: publishUpdates

3.1.14.1 General

This operation cleans the assertions table, publishes an update as specified by the parameter TestUpdatePublication and checks the header of the Publish message (see 3.1.4).

Operation Name	publishUpdates
Interaction Pattern	SEND

IP Sequence	Message	Field Type
IN	Send	TestUpdatePublication

3.1.15 OPERATION: publishError

3.1.15.1 General

This operation cleans the assertions table, publishes an update as specified by the parameter TestErrorPublication and checks the header of the Publish message (see 3.1.3).

Operation Name	publishError	
Interaction Pattern	SEND	
IP Sequence	Message	Field Type
IN	Send	TestErrorPublication

3.2 DataTest Service

This service aims at testing the data structures.

It provides an operation 'testData' that enables to transmit any Element to the provider and check that it is well interpreted by the provider.

Area Identifier	Service Identifier	Area Number	Service Number	Service Version
MALPrototype	DataTest	<test area nb>	0	1
Interaction Pattern	Operation Name	Operation Number	Support in replay	Capability Set
REQUEST	testData	100	No	100

3.2.1 OPERATION: testData

3.2.1.1 General

The 'testData' operation allows a consumer to check that a data is correctly decoded on the provider side. The provider needs to statically know the list of data that the consumer is going to send. The consumer selects the data in the same order as the list and calls the operation 'testData'. The provider keeps the index of the currently selected data from the static list. When the operation 'testData' is called, the provider checks that the received data is equal to the selected data from the list. If the equality test fails, then the error DATA_ERROR is raised otherwise the provider returns Null.

Operation Name	testData

Interaction Pattern	REQUEST	
IP Sequence	Message	Field Type
IN	Request	MAL::Element
OUT	Response	MAL::Element

3.2.1.2 Errors

The following error can be raised by this operation:

Error	Error #	Comments
DATA_ERROR	<DATA ERROR CODE>	Data interoperability error

3.3 ErrorTest Service

This service aims at testing the MAL errors. It doesn't need to be implemented as it is only used on the consumer side to make the transport layer raise errors (see section 4.3.1).

3.3.1 General

Area Identifier	Service Identifier	Area Number	Service Number	Service Version
MALPrototype	ErrorTest	<test area nb>	0	1
Interaction Pattern	**Operation Name**	**Operation Number**	**Support in replay**	**Capability Set**
REQUEST	testDeliveryFailed	100	No	100
REQUEST	testDeliveryTimedout	101	No	
REQUEST	testDeliveryDelayed	102	No	
REQUEST	testDestinationUnknown	103	No	
REQUEST	testDestinationTransient	104	No	
REQUEST	testDestinationLost	105	No	
REQUEST	testEncryptionFail	106	No	
REQUEST	testUnsupportedArea	107	No	
REQUEST	testUnsupportedOperation	108	No	

REQUEST	testUnsupportedVersion	109	No	100
REQUEST	testBadEncoding	110	No	
REQUEST	testUnknown	111	No	
REQUEST	testAuthenticationFailure	112	No	
REQUEST	testAuthorizationFailure	113	No	

3.3.2 OPERATION: testDeliveryFailed

3.3.2.1 General

This operation does nothing. Actually the error is raised by the transport layer before the provider is invoked.

Operation Name	testDeliveryFailed	
Interaction Pattern	REQUEST	
IP Sequence	Message	Field Type
IN	Request	MAL::Element
OUT	Response	MAL::Element

3.3.3 OPERATION: testDeliveryTimedout

3.3.3.1 General

This operation does nothing. Actually the error is raised by the transport layer before the provider is invoked.

Operation Name	testDeliveryTimedout	
Interaction Pattern	REQUEST	
IP Sequence	Message	Field Type
IN	Request	MAL::Element
OUT	Response	MAL::Element

3.3.4 OPERATION: testDeliveryDelayed

3.3.4.1 General

This operation does nothing. Actually the error is raised by the transport layer before the provider is invoked.

Operation Name	testDeliveryDelayed	
Interaction Pattern	REQUEST	
IP Sequence	Message	Field Type
IN	Request	MAL::Element
OUT	Response	MAL::Element

3.3.5 OPERATION: testDestinationUnknown

3.3.5.1 General

This operation does nothing. Actually the error is raised by the transport layer before the provider is invoked.

Operation Name	testDestinationUnknown	
Interaction Pattern	REQUEST	
IP Sequence	Message	Field Type
IN	Request	MAL::Element
OUT	Response	MAL::Element

3.3.6 OPERATION: testDestinationTransient

3.3.6.1 General

This operation does nothing. Actually the error is raised by the transport layer before the provider is invoked.

Operation Name	testDestinationTransient	
Interaction Pattern	REQUEST	
IP Sequence	Message	Field Type
IN	Request	MAL::Element

OUT	Response	MAL::Element

3.3.7 OPERATION: testDestinationLost

3.3.7.1 General

This operation does nothing. Actually the error is raised by the transport layer before the provider is invoked.

Operation Name	testDestinationLost	
Interaction Pattern	REQUEST	
IP Sequence	Message	Field Type
IN	Request	MAL::Element
OUT	Response	MAL::Element

3.3.8 OPERATION: testEncryptionFail

3.3.8.1 General

This operation does nothing. Actually the error is raised by the transport layer before the provider is invoked.

Operation Name	testEncryptionFail	
Interaction Pattern	REQUEST	
IP Sequence	Message	Field Type
IN	Request	MAL::Element
OUT	Response	MAL::Element

3.3.9 OPERATION: testUnsupportedArea

3.3.9.1 General

This operation does nothing. Actually the error is raised by the transport layer before the provider is invoked.

Operation Name	testUnsupportedArea	
Interaction Pattern	REQUEST	
IP Sequence	Message	Field Type

IN	Request	MAL::Element
OUT	Response	MAL::Element

3.3.10 OPERATION: testUnsupportedOperation

3.3.10.1 General

This operation does nothing. Actually the error is raised by the transport layer before the provider is invoked.

Operation Name	testUnsupportedOperation	
Interaction Pattern	REQUEST	
IP Sequence	Message	Field Type
IN	Request	MAL::Element
OUT	Response	MAL::Element

3.3.11 OPERATION: testUnsupportedVersion

3.3.11.1 General

This operation does nothing. Actually the error is raised by the transport layer before the provider is invoked.

Operation Name	testUnsupportedVersion	
Interaction Pattern	REQUEST	
IP Sequence	Message	Field Type
IN	Request	MAL::Element
OUT	Response	MAL::Element

3.3.12 OPERATION: testBadEncoding

3.3.12.1 General

This operation does nothing. Actually the error is raised by the transport layer before the provider is invoked.

Operation Name	testBadEncoding
Interaction Pattern	REQUEST

IP Sequence	Message	Field Type
IN	Request	MAL::Element
OUT	Response	MAL::Element

3.3.13 OPERATION: testUnknown

3.3.13.1 General

This operation does nothing. Actually the error is raised by the transport layer before the provider is invoked.

Operation Name	testUnknown	
Interaction Pattern	REQUEST	
IP Sequence	Message	Field Type
IN	Request	MAL::Element
OUT	Response	MAL::Element

3.3.14 OPERATION: testAuthenticationFailure

3.3.14.1 General

This operation does nothing. Actually the error is raised by the MAL layer before the provider is invoked.

Operation Name	testAuthenticationFailure	
Interaction Pattern	REQUEST	
IP Sequence	Message	Field Type
IN	Request	MAL::Element
OUT	Response	MAL::Element

3.3.15 OPERATION: testAuthorizationFailure

3.3.15.1 General

This operation does nothing. Actually the error is raised by the MAL layer before the provider is invoked.

Operation Name	testAuthorizationFailure
Interaction Pattern	REQUEST

IP Sequence	Message	Field Type
IN	Request	MAL::Element
OUT	Response	MAL::Element

4 Test scenarios

A scenario is comprised of several test cases. Each test case launch several test procedures.

Scenarios, test cases and procedures are uniquely identified by a name. The also have a status that can be RUN, DONE or FAIL.

All the scenarios are coordinated at the consumer side:

- They are started by a consumer initiating an interaction
- The status of scenarios, test cases and procedures is determined on the consumer side.

  Of course, assertions can be checked on both side: consumer and provider.

A test procedure is in charge of checking a set of assertions. There are two possible results for an assertion: OK (the assertion succeeded) or ERROR (the assertion failed).

A test procedure is DONE if it completes and if all the assertions are OK. If it is not completed, its status is RUN. If it is completed and if at least one assertion is in ERROR then its status is FAIL.

A test case is DONE if all the test procedures complete with the status DONE. If at least one procedure is not completed, its status is RUN. If all the procedures have completed and if at least one of them is FAIL then the test case status is FAIL.

A scenario is DONE if all the test cases complete with the status DONE. If at least one test case is not completed, its status is RUN. If all the test cases have completed and if at least one of them is FAIL then the scenario status is FAIL.

Providers are implemented as specified in section 3.

Two processes are launched:

1. The first one is called the “TestCoordinator” process. It launches the test procedures.
2. The second one is called the “TestPeer” process.

The “TestPeer” process instantiates the following providers:

- IPTest provider with a private broker
- IPTest provider with a shared broker
- DataTest provider without broker

It writes the URIs of providers and brokers into a properties file that is read by the TestCoordinator process.

The TestCoordinator process is launched after the TestPeer started in order that:

- The providers are ready to be called

- The URIs properties file is ready to be read by the TestCoordinator

4.1 IP test scenario

Two test cases are defined:

- The first one tests every interaction pattern except the Pub/Sub pattern.
- The second one is dedicated to the Pub/Sub pattern which is different as it does not involve the provider in the same way and it is more complex. Several aspects of the Pub/Sub pattern need to be tested. One test procedure is done for each of them.

Pub/Sub tests are to be done with a private and a shared broker.

Two constraints are required in order to check the IP state charts:

- no message loss
- FIFO message ordering is required

As a consequence, the Best Effort QoS can only be used if the specific transport layer ensures in the context of the test bed that messages are delivered exactly once and according to a FIFO ordering.

4.1.1 Test case: all patterns except Pub/Sub

The consumer initiates the patterns by calling the following operations provided by the service IPTest:

- send
- submit
- request
- invoke
- progress

Those operations shall be called once for each QoS level and session type. It is not necessary to test each combination of QoS and session. One call for each QoS level and session type is enough.

The following parameters are used to make the calls:

authenticationId	{0x00, 0x01}
qos	Best Effort, Assured, Queued, Timely
priority	1
domain	{“Test”, “Domain”}

networkZone	“TestNetwork”
session	Live, Simulation, Replay
session name	If the session type is Live, the name is “LIVE”. If the session type is Replay, the name is “R1”. If the session type is Simulation, the name is “S1”.

Moreover it is necessary to go through all the transitions of the IP state charts. The following IPTestDefinitions have to be instantiated. The table only gives the IPTestTransitionType part of the IPTestTransition. The faulty transitions are underlined. In the faulty case, the field 'errorCode' of the IPTestTransition is set to the value INCORRECT_STATE otherwise it is set to “-1”. The field 'Transition list id' is used to identify the test procedure.

IPTest operation	IPTestTransitionList	Transition list id
submit	{ACK}	1
	{ACK_ERROR}	2
	{ACK, ACK_ERROR}	3
	{ACK_ERROR, ACK_ERROR}	4
request	{RESPONSE}	1
	{RESPONSE_ERROR}	2
	{RESPONSE, RESPONSE}	3
	{RESPONSE_ERROR, RESPONSE}	4
invoke	{ACK, RESPONSE}	1
	{ACK, RESPONSE_ERROR}	2
	{ACK_ERROR}	3
	{ACK, RESPONSE, RESPONSE}	4
	{ACK, RESPONSE_ERROR, RESPONSE}	5
	{ACK_ERROR, ACK}	6
	{RESPONSE, ACK, RESPONSE}	7
progress	{ACK, RESPONSE}	1

	{ACK, RESPONSE_ERROR}	2
	{ACK_ERROR}	3
	{ACK, UPDATE, UPDATE, RESPONSE}	4
	{ACK, UPDATE, UPDATE, UPDATE_ERROR}	5
	{ACK, UPDATE, UPDATE, RESPONSE_ERROR}	6
	{ACK, RESPONSE, RESPONSE}	7
	{ACK, RESPONSE_ERROR, RESPONSE}	8
	{ACK_ERROR, ACK}	9
	{UPDATE, ACK, UPDATE, UPDATE, RESPONSE}	10
	{UPDATE_ERROR, ACK, UPDATE, UPDATE, RESPONSE}	11
	{RESPONSE, ACK, UPDATE, UPDATE, UPDATE_ERROR}	12
	{ACK, UPDATE, UPDATE, UPDATE_ERROR, RESPONSE}	13
	{ACK, UPDATE, UPDATE, RESPONSE_ERROR, RESPONSE}	14

The consumer has to execute the following test procedure for every possible header values (QoS and session fields) and every possible transitions. The name of the procedure is built from the parameters:

- ip: name of the tested IP
- qos: QoS level to be used
- session: session type to be used
- transition list id: identifier of the IPTestTransitionList that is expected by the consumer (see table above).

4.1.1.1 Test procedure: <ip>/<qos>/<session>/<transition list id>

The test procedure does the following actions:

1. Call the operation <ip> provided by the IPTest service with the parameters passed to the test

procedures: 'ip', 'qos', 'session' and 'transition list id'.

2. Wait for the expected transitions to be done, except the faulty ones that are ignored.
3. Call the operation 'getResult' provided by the IPTest service.
4. Check the provider assertions from the IPTestResult returned by 'getResult'
5. Check the message header
6. Check the transitions

a Check message header

When the consumer receives a MAL message it has to check that the header is the same as the expected header.

The expected header is deduced from the IPTestDefinition as follows:

Field	Assigned with
URIfrom	The provider's URI
authenticationId	Field 'authenticationId' of the IPTestDefinition.
URIto	Field 'consumerURI' of the IPTestDefinition.
timestamp	Field 'timestamp' of the IPTestDefinition.
QoSlevel	Field 'qos' of the IPTestDefinition.
priority	Field 'priority' of the IPTestDefinition.
domain	Field 'domain' of the IPTestDefinition.
networkZone	Field 'networkZone' of the IPTestDefinition.
session	Field 'session' of the IPTestDefinition.
sessionName	Field 'sessionName' of the IPTestDefinition.
interactionType	Interaction type used by the operation, e.g. the operation “send” uses a SEND interaction type.
interactionStage	Expected interaction stage as specified by the IPTestDefinition
transactionId	Assigned with the transaction identifier returned by the provider in the structure IPTestResult.
area	The test area name “MALPrototype”.
service	The test service name “IPTest”.
operation	The operation name.
version	The test service version.
isError	Depends on the expected interaction stage as specified by the IPTestDefinition:

	• True if a *_ERROR transition is expected • False otherwise

For each field, except 'timestamp', the following assertion is made:

Assertion	
Info	Result
Check header field '<field name>'	True if the value of the received header field is equal to the expected value fields False otherwise.

For 'timestamp' the assertion is:

Assertion	
Info	Result
Check header field 'timestamp'	True if the value of the received header timestamp is greater than the expected header timestamp. False otherwise.

b Check the transitions

The following assertion has to be checked for each transition of the IPTestTransitionList:

Assertion	
Info	Result
Check transition step #<transition index in the IPTestTransitionList>	True if the value of the received 'transition stage' is equal to the value of the expected stage (extracted from the IPTestTransitionList) False otherwise.

4.1.2 Test case: Pub/Sub interaction

This test checks that the message header is correct during the Pub/Sub interaction.

4.1.2.1 Test procedure: pubsub/<qos>/<session>

The test procedure is described below. It shall be executed once for each QoS level and session type. It is not necessary to test each combination of QoS and session. One execution for each QoS level and session type is enough.

The consumer calls the operation 'addPublishedEntities' with the entity key “A”.

The consumer creates subscriptions from the following parameters:

subscription id	“sub1”
authenticationId	{0x00, 0x01}
qos	Best Effort, Assured, Queued, Timely
priority	1
domain	{“Test”, “Domain”}
networkZone	“TestNetwork”
session	Live, Simulation, Replay
session name	If the session type is Live, the name is “LIVE”. If the session type is Replay, the name is “R1”. If the session type is Simulation, the name is “S1”.
entity expression	A
only on change	false

a Check Register

The Register header is obtained through the test transport module (see section 4.5).

The expected header is built as follows:

Field	Assigned with
URIfrom	The broker's URI
authenticationId	Parameter 'authenticationId'
URIto	The consumer's URI
timestamp	Current time before the subscription
QoSlevel	Parameter 'qos'
priority	Parameter 'priority'
domain	Parameter 'domain'
networkZone	Parameter 'networkZone'
session	Parameter 'session'
sessionName	Parameter 'sessionName'
interactionType	Pub/Sub
interactionStage	1
transactionId	Not assigned.

area	The test area name “MALPrototype”.
service	The test service name “IPTest”.
operation	The operation name.
version	The test service version.
isError	False

For each field, except 'timestamp' and 'transactionId', the following assertion is made:

Assertion	
Info	Result
Check header field '<field name>'	True if the value of the received header field is equal to the expected value fields False otherwise.

For 'timestamp' the assertion is:

Assertion	
Info	Result
Check header field 'timestamp'	True if the value of the received header timestamp is greater than the expected header timestamp. False otherwise.

For 'transactionId' the assertion is:

Assertion	
Info	Result
Check header field 'transactionId'	True if the value of 'transactionId' changes at each registration. False otherwise.

b Check Register acknowledgement

The expected header is built in the same way as before in section a except for the fields:

Field	Assigned with
interactionStage	2
transactionId	Field 'transactionId' of the Register message header.

For each field, except 'timestamp', the following assertion is made:

Assertion	
Info	Result

Check header field '<field name>'	True if the value of the received header field is equal to the expected value fields False otherwise.

For 'timestamp' the assertion is:

Assertion	
Info	Result
Check header field 'timestamp'	True if the value of the received header timestamp is greater than the expected header timestamp. False otherwise.

c Check Publish

The consumer does the following actions:

1. Call the operation 'publishUpdates' provided by the IPTest service. The consumer triggers publications with the entity key “A”. Four publications are passed as parameters of the call: one for each update type: creation, update, modification and deletion.

2. Call the operation 'getResult' provided by the IPTest service.

3. Check the provider assertions from the IPTestResult returned by 'getResult'

d Check Notify

The consumer checks the updates arrival (not implicit as the Notify reception is asynchronous):

Assertion	
Info	Result
Check Notify messages arrival	True if the four updates have been received. False otherwise.

The consumer checks the Notify header correctness. The expected header is built in the same way as before in section b except for the fields:

Field	Assigned with
timestamp	Current time before the update publications
interactionStage	4

The same assertions as in section b are done.

e Check Deregister

The consumer deregisters from the subscription “sub1”.

The Deregister header is obtained through the test transport module (see section 4.5).
The expected header is built in the same way as before in section b except for the fields:

Field	Assigned with
timestamp	Current time before the deregistration.
interactionStage	5

The same assertions as in section b are done.

f Check Deregister acknowledgement

The expected header is built in the same way as before in section b except for the fields:

Field	Assigned with
timestamp	Current time before the deregistration
interactionStage	6

The same assertions as in section b are done.

4.1.2.2 Test procedure: subscription checking

The consumer calls the operation 'addPublishedEntities' with the entity key “A” and “B”.

The consumer creates the subscription “sub1” defined in test procedure 4.1.2.1 with the QoS level Assured and the session LIVE.

The consumer keeps the transaction identifier of the registration.

The consumer redefines the subscription “sub1” (i.e. the consumer registers again) with two identical entity requests:

- same expression “A”
- same 'only on change' parameter set to the value 'true'.

The consumer triggers two publications in this order:

1. key = “A”, type = Update
2. key = “A”, type = Modification

The consumer checks:

Assertion	
Info	Result
Check update uniqueness	True if each update arrives only once despite the two entity requests own the same expression. False otherwise.
Check 'only on change'	True if the update which type is Update is not received ('only on change' is true). False otherwise.
Check the 'transactionId' invariance	True if 'transactionId' is the same as before the subscription redefinition.

4.1.2.3 Test procedure: publish error

The consumer creates another subscription as defined below:

subscription id	“sub2”
authenticationId	{0x00, 0x01}
qos	Assured
priority	1
domain	{“Test”, “Domain”}
networkZone	“TestNetwork”
session	Live
session name	“LIVE”
entity expression	B
only on change	false

a Check Publish error

The consumer does the following actions:

1. Call the operation 'publishError' provided by the IPTest service.
2. Call the operation 'getResult' provided by the IPTest service.
3. Check the provider assertions from the IPTestResult returned by 'getResult'

b Check Notify Error

The consumer triggers an error publication and checks:

Assertion	
Info	Result
Check error publication	True if the error is received by both subscriptions “sub1” and “sub2”. False otherwise.

The consumer checks the Notify Error header correctness. The expected header is built in the same way as above (see 4.1.2.1c), except for the field 'isError' which is 'true'.

Field	Assigned with
isError	True

The same assertions (see 4.1.2.1c) as above are done.

c Check subscription deletion

The subscription “sub2” must have been deleted by the published error. In order to check the subscription deletion, the consumer triggers two publications:

1. key = “A”, type = Modification
2. key = “B”, type = Modification

The consumer waits for a sufficient delay and checks:

Assertion	
Info	Result
Check subscription deletion	True if no update has arrived: the subscriptions have been deleted by the error publication. False otherwise.

4.1.2.4 Test procedure: subscription identifier uniqueness

This test procedure verifies that the URI of the consumer and the subscription identifier form the unique identifier of the subscription.

Two IPTest consumers are created. Two subscriptions with the same identifier “sub1” are created with both consumers as defined below:

Consumer URI	**URI #1**	**URI #2**
subscription id	“sub1”	“sub1”
authenticationId	{0x00, 0x01}	{0x00, 0x01}

qos	Assured	Assured
priority	1	1
domain	{“Test”, “Domain1”}	{“Test”, “Domain1”}
networkZone	“TestNetwork”	“TestNetwork”
session	Live	Live
session name	“LIVE”	“LIVE”
entity expression	“A”	“A”
only on change	false	false

The consumer triggers one publication with the key “A” and the update type “Modification”.

The consumer checks:

Assertion	
Info	Result
Check subscription identifier uniqueness	True if the update is received once by both consumers. False otherwise.

4.1.2.5 Test procedure: subscription entity requests

This test checks that the entity requests are correctly interpreted by the broker, in particular the expression used to define the expected entities.

A list of entity keys is defined:

1. A
2. A.B
3. A.B.C
4. A.B.C.D
5. B
6. Q.B.C

The consumer calls the operation 'addPublishedEntities' with the entity keys list presented above.

A list of entity request expressions is defined:

1. A
2. A.[null]
3. A.*

4. A.B.[null]
5. A.B.*
6. [null].B.[null]
7. *.B.*
8. *

The consumer does the following actions:

- Creation of one subscription for each expression and registration.

  The QoS level is Assured.

- Trigger the publication of one TestUpdate for each entity key.

  Use the same header values as in section 4.1.1.

Check that

Assertion	
Info	Result
Check entity request expression <expression>	True if the expected keys are received and only them (see table below). False otherwise.

The expected keys depending on the expression used are listed below:

Expression	**Expected keys**
A	A
A.[null]	A.B
A.*	A, A.B, A.B.C, A.B.C.D
A.B.[null]	A.B.C
A.B.*	A.B, A.B.C, A.B.C.D
[null].B.[null]	A.B.C, Q.B.C
.B.	A.B, A.B.C, A.B.C.D, B, Q.B.C
*	A, A.B, A.B.C, A.B.C.D, B, Q.B.C

4.1.2.6 Test procedure: registration error

The consumer registers to a private broker in order to receive the updates published by an unknown entity called “C”. The test procedure checks:

Assertion	
Info	Result
Check registration error	True if an error is raised. False otherwise.

4.1.2.7 Test procedure: deregistration error

The consumer deregisters from an unknown subscription “sub3”. The test procedure checks:

Assertion	
Info	Result
Check deregistration error	True if an error is raised. False otherwise.

4.1.2.8 Test procedure: notify error

The consumer creates the subscription “sub1” defined in test procedure 4.1.2.1 with the QoS level Assured and the session LIVE. Then the consumer deletes the provider and checks:

Assertion	
Info	Result
Check notify error	True if a Notify error is raised. False otherwise.

4.2 Data type test scenario

A list of MAL data structure instances is statically defined according to the following constraints:

- All the data types shall be instantiated at least once.
- Enumerations shall be instantiated once for each enumerated value.
- Abstract types need to be extended by a concrete type for the test
- The value Null shall belong to the list
- The value Null shall be inserted into a Composite structure

This data list is statically known by the DataTest service provider and consumer.

The consumer takes the data from the list one by one, in the same order, and calls the operation 'testData'. It checks that no error is raised by the provider, especially DATA_ERROR and BAD_ENCODING.

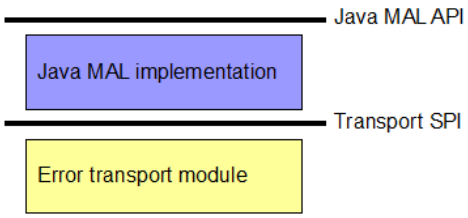
4.3 Error test scenario

The following tests use the ErrorTest service.

4.3.1 Test case: transport errors

These tests uses a specific transport module called “Error transport module” that raises errors depending on the operation called.

This transport module implements the Transport SPI as shown by the following figure:



Both primitives 'Transmit' and 'TransmitMultiple' return the following errors depending on the service and operation names:

Service	Operation	Error to raise
ErrorTest	testDeliveryFailed	DELIVERY_FAILED
ErrorTest	testDeliveryTimedout	DELIVERY_TIMEDOUT
ErrorTest	testDeliveryDelayed	DELIVERY_DELAYED
ErrorTest	testDestinationUnknown	DESTINATION_UNKNOWN
ErrorTest	testDestinationTransient	DESTINATION_TRANSIENT
ErrorTest	testDestinationLost	DESTINATION_LOST
ErrorTest	testEncryptionFail	ENCRYPTION_FAIL
ErrorTest	testUnsupportedArea	UNSUPPORTED_AREA
ErrorTest	testUnsupportedOperation	UNSUPPORTED_OPERATION
ErrorTest	testUnsupportedVersion	UNSUPPORTED_VERSION
ErrorTest	testBadEncoding	BAD_ENCODING
ErrorTest	testUnknown	UNKNOWN
IPTest	monitor	ENCRYPTION_FAIL (used in section 4.5.4.2)

4.3.1.1 Test procedure: Delivery failed

The ErrorTest consumer calls the operation testDeliveryFailed and checks that the error DELIVERY_FAILED is returned.

Assertion	
Info	Result
Check DELIVERY_FAILED error	True if a DELIVERY_FAILED error is raised. False otherwise.

4.3.1.2 Test procedure:Delivery timedout

The ErrorTest consumer calls the operation testDeliveryTimedout and checks that the error DELIVERY_TIMEDOUT is returned.

Assertion	
Info	Result
Check DELIVERY_TIMEDOUT error	True if a DELIVERY_TIMEDOUT error is raised. False otherwise.

4.3.1.3 Test procedure:Delivery delayed

The ErrorTest consumer calls the operation testDeliveryDelayed and checks that the error DELIVERY_DELAYED is returned.

Assertion	
Info	Result
Check DELIVERY_DELAYED error	True if a DELIVERY_DELAYED error is raised. False otherwise.

4.3.1.4 Test procedure:Destination unknown

The ErrorTest consumer calls the operation testDestinationUnknown and checks that the error DESTINATION_UNKNOWN is returned.

Assertion	
Info	Result
Check DESTINATION_UNKNOWN error	True if a DESTINATION_UNKNOWN error is raised. False otherwise.

4.3.1.5 Test procedure: Destination transient

The ErrorTest consumer calls the operation testDestinationTransient and checks that the error DESTINATION_TRANSIENT is returned.

Assertion	
Info	Result
Check DESTINATION_TRANSIENT error	True if a DESTINATION_TRANSIENT error is raised. False otherwise.

4.3.1.6 Test procedure: Destination lost

The ErrorTest consumer calls the operation testDestinationLost and checks that the error DESTINATION_LOST is returned.

Assertion	
Info	Result
Check DESTINATION_LOST error	True if a DESTINATION_LOST error is raised. False otherwise.

4.3.1.7 Test procedure: Encryption fail

The ErrorTest consumer calls the operation testEncryptionFail and checks that the error ENCRYPTION_FAIL is returned.

Assertion	
Info	Result
Check ENCRYPTION_FAIL error	True if a ENCRYPTION_FAIL error is raised. False otherwise.

4.3.1.8 Test procedure: Unsupported area

The ErrorTest consumer calls the operation testUnsupportedArea and checks that the error UNSUPPORTED_AREA is returned.

Assertion	
Info	Result
Check UNSUPPORTED_AREA error	True if a UNSUPPORTED_AREA error is raised. False otherwise.

4.3.1.9 Test procedure: Unsupported operation

The ErrorTest consumer calls the operation testUnsupportedOperation and checks that the error UNSUPPORTED_OPERATION is returned.

Assertion	
Info	Result
Check UNSUPPORTED_OPERATION error	True if a UNSUPPORTED_OPERATION error is raised. False otherwise.

4.3.1.10 Test procedure: Unsupported version

The ErrorTest consumer calls the operation testUnsupportedVersion and checks that the error UNSUPPORTED_VERSION is returned.

Assertion	
Info	Result
Check UNSUPPORTED_VERSION error	True if a UNSUPPORTED_VERSION error is raised. False otherwise.

4.3.1.11 Test procedure: Bad encoding

The ErrorTest consumer calls the operation testBadEncoding and checks that the error BAD_ENCODING is returned.

Assertion	
Info	Result
Check BAD_ENCODING error	True if a BAD_ENCODING error is raised. False otherwise.

4.3.1.12 Test procedure: Unknown

The ErrorTest consumer calls the operation testUnknown and checks that the error UNKNOWN is returned.

Assertion	
Info	Result
Check UNKNOWN error	True if a UNKNOWN error is raised. False otherwise.

4.3.2 Test case: security errors

A test security module is to be implemented in order to raise errors depending on the operation called:

Service	Operation	Error to raise
ErrorTest	testAuthenticationFailure	AUTHENTICATION_FAIL
ErrorTest	testAuthorizationFailure	AUTHORIZATION_FAIL

4.3.2.1 Test procedure: authentication failure

The ErrorTest consumer calls the operation testAuthenticationFail and checks that the error AUTHENTICATION_FAIL is returned.

Assertion	
Info	Result
Check AUTHENTICATION_FAIL error	True if a AUTHENTICATION_FAIL error is raised. False otherwise.

4.3.2.2 Test procedure: authorization failure

The ErrorTest consumer calls the operation testAuthorizationFail and checks that the error AUTHORIZATION_FAIL is returned.

Assertion	
Info	Result
Check AUTHORIZATION_FAIL error	True if a AUTHORIZATION_FAIL error is raised. False otherwise.

4.4 Access control test scenario

The Access Control interface has to be implemented by a test security module in charge of checking that the primitive CHECK is called by the MAL. The message passed with the CHECK request is returned to the MAL without being modified through the CHECK Response indication.

The consumer process has to create a MAL instance that uses the test security module.

4.4.1 Test case: CHECK interaction

4.4.1.1 Test procedure: CHECK interaction

The consumer has to do the following actions:

1. create an IPTest consumer
2. initiate a Request interaction by calling the operation “request”
3. wait for the end of the interaction
4. check that the MAL has interacted twice with the Access Control. The following assertion is checked:

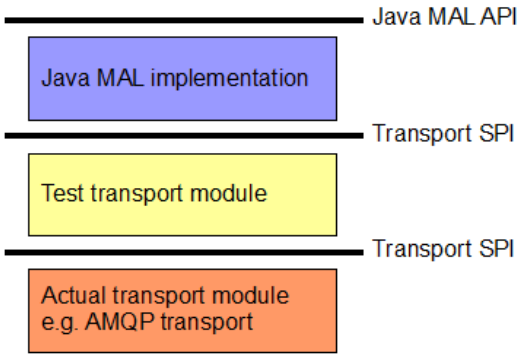
Assertion	
Info	Result
Check the CHECK interaction for a message coming from the application and for a message coming from the transport	True if the Access Control primitive "CHECK Request" has been called twice: • once for a message which interaction stage is "1" (Request coming from the application) • and a second time for a message which interaction stage is "2" (Request response coming from the transport). False otherwise.

4.5 Transport test scenario

The Transport interface is implemented by a test transport module in charge of:

- Checking that the Transport primitives are called by the MAL
- Keeping the last message that has been transmitted to the transport by the MAL (this feature is used by the test procedure 4.1.2.1)
- Forwarding the primitive Request to the actual transport layer
- Listening to the Indications triggered by the actual transport and transmitting them to the MAL

This test transport module is an intermediate layer between the MAL and the actual transport used for the test as shown by the following figure:



4.5.1 Test case: supported QoS interaction

4.5.1.1 Test procedure: supported QoS interaction

An IPTest consumer is created for each available QoS level: BEST_EFFORT, ASSURED, QUEUED and TIMELY. It is important that the MAL instance was not used before as the MAL only calls the SUPPORTEDIP once when first interacting with a specific transport.

The test checks the following assertion:

Assertion	
Info	Result
Check the SUPPORTEDQOS interaction.	True if: • the primitive 'SUPPORTEDQOS Request' has been called once by the MAL for each QoS level • and the value of the QoSLevel parameter is equal to the QoSLevel used when creating the consumer • and the primitive 'SUPPORTEDQOS Indication' has been called once by the actual transport for each QoS level. False otherwise.

4.5.2 Test case: supported IP interaction

4.5.2.1 Test procedure: supported IP interaction

A single instance of MAL and an IPTest consumer are created. It is important that the MAL instance was not used before as the MAL only calls the SUPPORTEDIP once when first interacting with a specific transport.

The test checks the following assertion:

Assertion	
Info	Result
Check the SUPPORTEDIP interaction.	True if: • the primitive 'SUPPORTEDIP Request' has been called once by the MAL for each InteractionType. • and the primitive 'SUPPORTEDQIP Indication' has been called once by the actual transport for each InteractionType. False otherwise.

4.5.3 Test case: transmit interaction

4.5.3.1 Test procedure: transmit interaction

An IPTest consumer is created and a Send interaction is initiated by calling the operation “send”.

The test checks the following assertion:

Assertion	
Info	Result
Check the TRANSMIT interaction.	True if: • the primitive 'TRANSMIT Request' has been called once by the MAL with a message which interaction stage is “1” (Send coming from the application) • and the primitive 'TRANSMIT Indication' has been called once by the actual transport. False otherwise.

4.5.4 Test case: transmit multiple interaction

4.5.4.1 Test procedure: transmit multiple interaction

This is a single process test: 'TestCoordinator'. This test doesn't check the MAL interoperability so it is not necessary to have two processes with two MAL implementations.

An IPTest provider and a private broker are created. Two IPTest consumers are created and registered to the private broker through the operation 'monitor. An update is published by calling the operation 'publishUpdates'.

The test checks the following assertion:

Assertion	
Info	Result
Check the TRANSMITMULTIPLE interaction.	True if: • the primitive 'TRANSMITMULTIPLE Request' has been called once with two messages which interaction stage is “3” (Publish coming from the application) • and the primitive 'TRANSMITMULTIPLE Indication' has been called once by the actual transport. False otherwise.

4.5.4.2 Test procedure: transmit multiple error

The same test as above (see 4.5.4.1) is launched on top of the “Error transport module” (see section 4.3.1).

The test checks the following assertion:

Assertion	
Info	Result
Check the TRANSMITMULTIPLE Error indication.	True if the error ENCRYPTION_FAIL is caught by the publisher. False otherwise.

4.5.5 Test case: receive interaction

4.5.5.1 Test procedure: receive interaction

An IPTest consumer is created and a Request interaction is initiated by calling the operation “request”.

The test checks the following assertion:

Assertion	
Info	Result
Check the RECEIVE interaction.	True if the primitive 'TRANSMIT Request' has been called once by the MAL with a message which interaction stage is “2” (Request response coming from the actual transport) False otherwise.

4.5.6 Test case: receive multiple interaction

4.5.6.1 Test procedure: receive multiple interaction

As it is not possible to make the assumption that a transport module uses this interaction, the test transport module has to be enhanced with an operation enabling to trigger a RECEIVEMULTIPLE Indication.

An IPTest consumer is created. It is registered to the IPTest provider private broker running in the “TestPeer” process. One update is published by calling the operation “publishUpdate”. The Notify message is received by the consumer. Then the test procedure copies this message twice, making two Notify messages and directly injects them into the test transport module in order to trigger a RECEIVEMULTIPLE Indication.

The test checks the following assertion:

Assertion	
Info	Result
Check the RECEIVEMULTIPLE interaction.	True if the two Notify messages are received by the consumer. False otherwise.

5 MALPrototype Data Types

This section defines the data types used by the test services (see section 3).

5.1 Data Structures

5.1.1 Assertion

Structure Name	Assertion	
Extends	MAL::Composite	
Short form	test_asrt	
Field	Type	Comment
procedureName	MAL::String	Name of the test procedure that evaluated the assertion.
info	MAL::String	Message explaining what the assertion checks.
result	MAL::Boolean	Boolean indicating whether the assertion succeeded (true) or not (false).

5.1.2 AssertionList

List Name	AssertionList
Short form	test_asrt_lst
List of	Assertion

5.2 IPTest Service Structures

5.2.1 IPTestDefinition

This abstract structure is inherited by all the IP test definition structures.

Structure Name	IPTestDefinition	
Extends	MAL::Composite	
Short form	test_ip_def	
Field	Type	Comment

procedureName	MAL::String	Name of the test procedure
consumerURI	MAL::URI	The consumer's URI
authenticationId	MAL:Blob	The authentication identifier used by the consumer
qos	MAL::QoSLevel	The QoS level required by the consumer
priority	MAL::Integer	The priority level required by the consumer
domain	MAL::DomainIdentifier	The domain used by the consumer
networkZone	MAL::Identifier	The network zone used by the consumer
session	MAL::SessionType	The type of the session used by the consumer
sessionName	MAL::Identifier	The identifier of the session used by the consumer
transitions	IPTestTransitionList	The transitions that are requested by the consumer
timestamp	MAL::Date	The time the consumer initiated the interaction.

5.2.2 IPTestTransitionType

This enumeration is used to require an expected transition from an IP test.

Enumeration Name	IPTestTransitionType	
Short form	test_ip_trt	
Enumeration Value	Short form	Comment
ACK	1	Return an acknowledgement.
RESPONSE	2	Return a response.
ACK_ERROR	3	Return an error as an acknowledgement. The error is a TEST_ERROR.
RESPONSE_ERROR		Return an error as a response. The error is a TEST_ERROR.

UPDATE	4	Return a progress update.
UPDATE_ERROR	5	Return an error as an update. The error is a TEST_ERROR.

5.2.3 IPTestTransition

This structure is used to define an expected transition from an IP test. It asserts what transition is expected and what result is expected from the transition: successful or failure.

Structure Name	IPTestTransition	
Extends	MAL::Composite	
Short form	test_ip_tr	
Field	Type	Comment
type	IPTestTransitionType	The type of the transition to do
errorCode	MAL::Integer	The code of the error expected to be raised when doing the transition (failed transition). -1 if no error is expected (successful transition).

5.2.4 IPTestTransitionList

List Name	IPTestTransitionTypeList
Short form	test_ip_trl
List of	IPTestTransition

5.2.5 BadHeaderReport

This data structure is an error report produced after having found a faulty header.

Structure Name	BadHeaderReport	
Extends	MAL::Composite	
Short form	test_ip_bhr	
Field	Type	Comment
expectedHeader	MAL::MessageHeader	The expected header

faultyHeader	MAL::MessageHeader	The header that is not compliant with the MAL rules

5.2.6 BadHeaderReportList

This data structure is a list of BadHeaderReport.

List Name	BadHeaderReportList
Short form	test_ip_bhrl
List of	BadHeaderReport

5.2.7 TestPublication

This abstract structure is inherited by all the update and error publication structures.

Structure Name	TestPublication	
Extends	MAL::Composite	
Abstract		
Field	Type	Comment
authenticationId	MAL:Blob	The authentication identifier used by the consumer
qos	MAL::QoSLevel	The QoS level to be used by the provider for publishing the update.
priority	MAL::Integer	The priority to be used by the provider for publishing the update.
domain	MAL::DomainIdentifier	The domain to be used by the provider for publishing the update.
networkZone	MAL::Identifier	The network zone to be used by the provider for publishing the update.
session	MAL::SessionType	The session type to be used by the provider for publishing the update.
sessionName	MAL::Identifier	The session name to be used by the provider for publishing the update.

5.2.8 TestUpdatePublication

This data structure specifies how the IPTest provider shall publish an update.

Structure Name	TestUpdatePublication	
Extends	TestPublication	
Short form	test_ip_tup	
Field	Type	Comment
update	MAL::UpdateList	The updates to be published by the provider

5.2.9 TestUpdate

This data structure defines an Update published by the IPTest.

Structure Name	TestUpdate	
Extends	MAL::Update	
Short form	test_ip_tu	
Field	Type	Comment
counter	MAL::Integer	A counter used to distinguish the test updates and to check the ordering.

5.2.10 TestErrorPublication

This data structure specifies how the IPTest provider shall publish an error.

Structure Name	TestErrorPublication	
Extends	TestPublication	
Short form	test_ip_tep	
Field	Type	Comment
error	MAL::StandardError	The error to be published by the provider

5.2.11 InteractionKey

Structure Name	InteractionKey	
Extends	MAL::Composite	
Short form	test_ip_itrk	
Field	Type	Comment

URIfrom	MAL::URI	The consumer's URI
transactionId	MAL::Integer	The transaction identifier of the interaction
interactionType	MAL::InteractionType	The type of the interaction
service	MAL::Identifier	The name of the called service
operation	MAL::Identifier	The name of the called operation

5.2.12 IPTestResult

Structure Name	IPTestResult	
Extends	MAL::Composite	
Short form	test_ip_tstr	
Field	Type	Comment
transactionId	MAL::Integer	The transaction identifier assigned to the last interaction
assertions	AssertionList	The list of assertions checked by the provider.

6 MAL book requirements

This section lists all the requirements specified in the MAL book that are checked by the test scenarios. Some requirements are implicitly tested by the API itself which restricts what a MAL client can do.

The tables below gather the MAL requirements and indicate either the test procedure responsible for checking it or how the Java MAL API implicitly verifies it.

6.1 Message Abstraction Layer

6.1.1 IP and service interface

Requirement		**Test procedure**
Transaction handling (4.2)		4.1.1.1 Test procedure: <ip>/<qos>/<session>/<transition list id> Assertion defined on the provider side: 3.1.1 Check message header (field 'transaction id')
		4.1.2.1 Test procedure: pubsub/<qos>/<session> Assertion defined in 4.1.2.1a Check Register ('transactionId' checking)
Error handling (4.4.x.4)	Submit (x=2), Request (x=3), Invoke (x=4), Progress (x=5)	4.1.1.1 Test procedure: <ip>/<qos>/<session>/<transition list id> 4.1.1.1a Check message header (field 'isError')
	Send (x=1)	Java MAL API: no error can be returned by a provider during a Send interaction.
	Pub/Sub registration (x=6)	4.1.2.6 Test procedure: registration error
	Pub/Sub publish (x=6)	4.1.2.3 Test procedure: publish error
	Pub/Sub notify (x=6)	4.1.2.8 Test procedure: notify error
	Pub/Sub deregistration (x=6)	4.1.2.7 Test procedure: deregistration error
Operation template and primitives (4.4.x.5, 4.4.x.6)		Java MAL API: the consumer and provider interfaces enable to send the Requests and receive the Indications defined for each IP.
Requests and Indications (4.4.x.8)	Send (x=1) Submit (x=2) Request (x=3) Invoke (x=4) Progress (x=5)	4.1.1.1 Test procedure: <ip>/<qos>/<session>/<transition list id> Assertions defined on the provider side: 3.1.1 Check message header

	SubmitAck (x=2) SubmitError (x=2) RequestResponse (x=3) RequestError (x=3) InvokeAck (x=4) InvokeAckError (x=4) InvokeResponse (x=4) InvokeResponseError (x=4) ProgressAck (x=5) ProgressAckError (x=5) ProgressUpdate (x=5) ProgressUpdateError (x=5) ProgressResponse (x=5) ProgressResponseError (x=5)	4.1.1.1 Test procedure: <ip>/<qos>/<session>/<transition list id> 4.1.1.1a Check message header
	PS Register Ack (4.4.6.11) PS Deregister Ack (4.4.6.11) PS Notify (4.4.6.11)	4.1.2.1 Test procedure: pubsub/<qos>/<session>
	PS Register (4.4.6.11) PS Deregister (4.4.6.11) PS Publish (4.4.6.11)	4.1.2.1 Test procedure: pubsub/<qos>/<session> 4.1.2.1a Check Register 4.1.2.1c Check Publish 4.1.2.1e Check Deregister
Pub/Sub specific features	Overview: unique subscription identifier (4.4.6.1)	4.1.2.4 Test procedure: subscription identifier uniqueness
	Description (4.4.6.2)	4.1.2.2 Test procedure: subscription checking
	Entity key (4.4.6.3)	4.1.2.5 Test procedure: subscription entity requests
State Charts (4.4.x.7)	Send (x=1) Submit (x=2) Request (x=3) Invoke (x=4) Progress (x=5)	4.1.1.1 Test procedure: <ip>/<qos>/<session>/<transition list id> 4.1.1.1b Check the transitions
	Pub/Sub: Consumer side (4.4.6.8.1)	4.1.2.1a Check Register 4.1.2.1b Check Register acknowledgement 4.1.2.1d Check Notify 4.1.2.1e Check Deregister 4.1.2.1f Check Deregister acknowledgement
		4.1.2.6 Test procedure: registration error
		4.1.2.7 Test procedure: deregistration error
		4.1.2.3b Check Notify Error 4.1.2.3c Check subscription deletion

	Pub/Sub: Provider side (4.4.6.8.2)	4.1.2.1c Check Publish
		4.1.2.3a Check Publish error

6.1.2 Access control interface

Requirement		Test procedure
Check interaction (4.5.2)	Check Check response	4.4.1.1 Test procedure: CHECK interaction
	Check error	4.3.2.1 Test procedure: authentication failure

6.1.3 Transport interface

Requirement		Test procedure
SupportedQoS interaction (4.6.2)		4.5.1.1 Test procedure: supported QoS interaction
SupportedIP interaction (4.6.3)		4.5.2.1 Test procedure: supported IP interaction
Transmit interaction (4.6.4)	Request and indication	4.5.3.1 Test procedure: transmit interaction
	Error indication	4.3.1.7 Test procedure: Encryption fail An error is raised by the transport layer during a transmit interaction.
Transmit multiple interaction (4.6.5)	Request and indication	4.5.4.1 Test procedure: transmit multiple interaction
	Error indication	4.5.4.2 Test procedure: transmit multiple error
Receive interaction (4.6.6)	Indication	4.5.5.1 Test procedure: receive interaction
Receive multiple interaction (4.6.7)	Indication	4.5.6.1 Test procedure: receive multiple interaction

6.2 Data types

All the data types are checked in section 4.2.

6.3 Errors

Requirement (5.2)	Test procedure
DELIVERY_FAILED	4.3.1.1 Test procedure: Delivery failed
DELIVERY_TIMEDOUT	4.3.1.2 Test procedure:Delivery timedout
DELIVERY_DELAYED	4.3.1.3 Test procedure:Delivery delayed
DESTINATION_UNKNWON	4.3.1.4 Test procedure:Destination unknown
DESTINATION_TRANSIENT	4.3.1.5 Test procedure: Destination transient

DESTINATION_LOST	4.3.1.6 Test procedure: Destination lost
AUTHENTICATION_FAIL	4.3.2.1 Test procedure: authentication failure
AUTHORIZATION_FAIL	4.3.2.2 Test procedure: authorization failure
ENCRYPTION_FAIL	4.3.1.7 Test procedure: Encryption fail
UNSUPPORTED_AREA	4.3.1.8 Test procedure: Unsupported area
UNSUPPORTED_OPERATION	4.3.1.9 Test procedure: Unsupported operation
UNSUPPORTED_VERSION	4.3.1.10 Test procedure: Unsupported version
BAD_ENCODING	4.3.1.11 Test procedure: Bad encoding
UNKNOWN	4.3.1.12 Test procedure: Unknown
INCORRECT_STATE	4.1.1.1 Test procedure: <ip>/<qos>/<session>/<transition list id> Check faulty transitions on the provider side (see 3.1.2)

7 Result output format

This section presents a template of output file to be generated by the MAL prototype:

```
MALPrototype                                                    RUN
|- Test Scenario <name>                                         RUN
  |- Test Case <name>                                           RUN
    |- Test Procedure <name>                                    RUN
      |- Assertion <info>                                       <OK|ERROR>
    |- Test Procedure <name>                                    <DONE|FAIL>
  |- Test Case <name>                                           <DONE|FAIL>
|- Test Scenario <name>                                         <DONE|FAIL>
MALPrototype                                                    <DONE|FAIL>
```