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.

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

The sixth part states what are the software modules to develop, by who and by when.

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 layer MAL/Joram is shared by both MAL implementations.

1.4 Test framework

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

If a single process can be used to launch both consumer and provider then the Ant script just has to start the tests in a sequential way.

If two processes are used for each test, Ant is responsible for starting them as parallel tasks. Then the consumer part of the application drives the test:

- resolve the provider's URIs
- invoke the operations
- execute the JUnit assertions
- stop the provider's process

2 Test cases overview

The test cases 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 URI format shall be: <scheme name> : <hierarchical part> [? <query>] [# <fragment>]
- 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 Standard error

2.3.1 Communication

Some communication errors are to be checked:

- unknown destination
- transient destination
- delivery timed-out: a consumer initiates an interaction with the QoS level Queued and a TTL. The provider is stopped so it cannot respond.

However 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.

2.3.2 Security

A security module is implemented for the test in order to raise the authentication and authorization errors.

The Login service is not required by this test. This is a MAL level limited test.

2.3.3 Encryption

The encryption error is tested by modifying a data structure generated from the service definition XML.

The decryption error (bad encoding) can be tested by making the consumer send a data structure which class is unknown on the provider side.

2.3.4 Area, operation, version

A service provider is implemented for the test in order to raise those errors.

2.4 QoS

The four levels of QoS shall be tested in order to check that they are correctly interpreted on both sides as a message header field. But it is difficult to really test the QoS for two reasons:

- The MAL is transport agnostic so it cannot precisely define a QoS level.

  For example, the interpretation of “ensures delivery of messages to its destination” requires more details at the transport level. Is it a simple connection based protocol (like TCP) ? Is it a fail-over protocol (with recovery after a connection failure) ? Is it a transactional protocol (with recovery after a client failure) ?

- Testing a QoS level leads to test the transport layer, not the MAL.

However, some tests are possible to implement:

- About the Queued QoS level: the time coupling can be tested by stopping/restarting a MAL client (provider or consumer) and checking that the messages are received by the client.
- The QoS property “TTL” can also be tested.
- The message ordering can be tested.

2.5 Priority

A scenario shall be defined in order to test the message priority feature on top of the MAL layer.

3 Test services

3.1 IPTest

This service aims at testing the Interaction Patterns.

It provides one operation per IP. The input parameter inherits from IPTestDefinition. It specifies how the provider is expected to behave (e.g. return an error) by the consumer. Moreover it contains the parameter values used by the consumer for configuring the interaction (e.g. the domain identifier).

For each operation (except the Pub/Sub one), the provider shall check that the received message header is compliant with the MAL rules. If it is not, it creates an instance of BadHeaderReport and adds it to the internal BadHeaderReportList.

The following assertions shall be checked in order to ensure that the message header is correct:

Field	Assertion
URIfrom	Checks the equality with the consumer's URI. Checks the URI format.
authenticationId	Checks the equality with the Blob used for the test.
URIto	Checks the equality with the provider's URI.
timestamp	Checks that the stamp is greater than the test timestamp (see IPTestDefinition) and less than the test date added to the test timeout. This checks that the header is assigned with an approximately correct date.
QoSlevel	Checks the equality with the QoS level used for the test.
priority	Checks the equality with the priority used for the test.
domain	Checks the equality with the domain used for the test.
networkZone	Checks the equality with the networkZone used for the test.
session	Checks the equality with the session type used for the test.
sessionName	Shall be 'LIVE' if session type is LIVE. Otherwise checks the equality with the session name used for the test.
interactionType	Checks the equality with the expected interaction type.
interactionStage	Checks the equality with the expected interaction stage.
transactionId	Nothing to check.
area	Checks the equality with the expected area.
service	Checks the equality with the expected service.
operation	Checks the equality with the expected operation.
version	Checks the equality with the expected service version.
isError	Checks that the message body is compliant with the value of this header.

An operation called 'getBadHeaderReports' enables the consumer to know if some inconsistencies in the

headers have been found by the provider.

Two operations 'publishUpdate' and 'publishError' enable the consumer to trigger an update or error publishing at the provider side.

The service interface is described below:

Area Identifier	Service Identifier	Area Number	Service Number	Service Version
<test area name>	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	getBadHeaderReports	106	No	101
SEND	addPublishedEntity	107	No	102
SEND	publishUpdate	108	No	103
SEND	publishError	109	No	
SEND	stop	110	No	104

3.1.1 send

This operation tests the IP “Send”.

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

3.1.2 submit

This operation initiates a Request interaction. The interaction shall be handled as specified by the SubmitTestDefinition parameter.

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

The following error can be raised by this operation:

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

3.1.3 request

This operation initiates a Request interaction. The interaction shall be handled as specified by the RequestTestDefinition parameter.

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

The following error can be raised by this operation:

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

3.1.4 invoke

This operation initiates an Invoke interaction. The interaction shall be handled as specified by the InvokeTestDefinition parameter.

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

The following error can be raised by this operation:

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

3.1.5 progress

This operation initiates a Progress interaction. The interaction shall be handled as specified by the ProgressTestDefinition parameter.

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

The following error can be raised by this operation:

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

3.1.6 monitor

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

3.1.7 addPublishedEntities

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

3.1.8 getBadHeaderReport

This operation returns the BadHeaderReportList filled by the provider. No input parameter is expected.

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

3.1.9 publishUpdate

This operation publishes an update as specified by the parameter TestUpdatePublication.

Operation Name	publishUpdate	
Interaction Pattern	SEND	
IP Sequence	Message	Field Type
IN	Send	TestUpdatePublication

3.1.10 publishError

This operation publishes an error as specified by the parameter TestErrorPublication.

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

3.1.11 Stop

This operation stops the provider's process. No parameter is expected.

Operation Name	stop	
Interaction Pattern	SEND	
IP Sequence	Message	Field Type
IN	Send	MAL::Element

3.2 DataTest

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
<test area name>	DataTest	<test area nb>	0	1
Interaction Pattern	**Operation Name**	**Operation Number**	**Support in replay**	**Capability Set**
REQUEST	testData	100	No	100
SEND	Stop	101	No	101

3.2.1 testData

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

The following error can be raised by this operation:

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

3.2.2 Stop

This operation stops the provider's process. No parameter is expected.

Operation Name	stop
Interaction Pattern	SEND

IP Sequence	Message	Field Type
IN	Send	MAL::Element

3.3 ErrorTest

This service aims at testing the MAL errors.

It provides several operations enabling a consumer to make the provider raise an error.

Area Identifier	Service Identifier	Area Number	Service Number	Service Version
<test area name>	ErrorTest	<test area nb>	0	1
Interaction Pattern	**Operation Name**	**Operation Number**	**Support in replay**	**Capability Set**
REQUEST	test	100	No	100
REQUEST	testEncryption	101	No	
REQUEST	raiseAreaError	102	No	
REQUEST	raiseOperationError	103	No	
REQUEST	raiseVersionError	104	No	
SEND	Stop	105	No	101

3.3.1 test

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

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

3.3.2 testEncryption

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

Operation Name	testEncryption

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

3.3.3 raiseAreaError

This operation shall raise an UNSUPPORTED_AREA error. No input parameter is expected.

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

3.3.4 raiseOperationError

This operation shall raise an UNSUPPORTED_OPERATION error.

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

3.3.5 raiseVersionError

This operation shall raise an UNSUPPORTED_VERSION error.

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

3.3.6 stop

This operation stops the provider's process. No parameter is expected.

Operation Name	stop	
Interaction Pattern	SEND	
IP Sequence	Message	Field Type
IN	Send	MAL::Element

3.4 PriorityTest

This service aims at testing the MAL priority.

The operation 'invokeAction' allows the consumer to check that the MAL priority is taken into account.

Area Identifier	Service Identifier	Area Number	Service Number	Service Version
<test area name>	PriorityTest	<test area nb>	0	1
Interaction Pattern	Operation Name	Operation Number	Support in replay	Capability Set
REQUEST	invokeAction	100	No	100
SEND	Stop	105	No	101

3.4.1 InvokeAction

This operation does the following actions:

1. sleep during the delay specified by the input parameter
2. increment an internal counter
3. return the value of the counter as a response

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

3.4.2 stop

This operation stops the provider's process. No parameter is expected.

Operation Name	stop	
Interaction Pattern	SEND	
IP Sequence	Message	Field Type
IN	Send	Element

4 Test scenarios

All the scenarios are coordinated by the consumer side. The consumer is responsible for:

- Initiating the interactions
- Checking the required assertions
- Stopping the provider process

Providers are implemented as specified in section 3.

The URIs of providers and brokers are statically known and given to the consumers through Java environment properties at start-up.

4.1 IPTest

Two scenarios are actually 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 is done for each of them:

- Message header consistency
- Interaction protocol checking
- Subscription name space
- Entity request correctness
- Unknown entity error

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

4.1.1 All patterns except Pub/Sub

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

1. send
2. submit
3. request
4. invoke
5. 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.

Notice that using the QoS Best Effort may lead the test to fail as a message can be lost (depending on the transport).

The call shall be done with the following values.

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 shall be instantiated:

IPTestDefinition	Constructor parameter values
SubmitTestDefinition	ack = true ack = false
RequestTestDefinition	res = true res = false
InvokeTestDefinition	ack = true, res = true ack = true, res = false ack = false res = false (res is not taken into account)
ProgressTestDefinition	ack = true, stepNumber = 0, res = true ack = true, stepNumber = 0, res = false ack = false, stepNumber = 0, res = false (stepNumber and res are not taken into account) ack = true, stepNumber = 2, res = true

During the interaction the consumer shall:

- check that the headers of the messages returned by the provider are correct. The following assertions are checked in order to ensure that the message header is correct:

Field	Assertion
URIfrom	Checks the equality with the provider's URI. Checks the URI format.
authenticationId	Checks the equality with the Blob used for the test.
URIto	Checks the equality with the consumer's URI.
timestamp	Checks that the stamp is greater than the test timestamp (see TestDefinition) and less than the test date added to the test timeout. This checks that the header is assigned with an approximately correct date.
QoSlevel	Checks the equality with the QoS level used for the test.
priority	Checks the equality with the priority used for the test.
domain	Checks the equality with the domain used for the test.
networkZone	Checks the equality with the networkZone used for the test.
session	Checks the equality with the session type used for the test.
sessionName	Shall be 'LIVE' if session type is LIVE. Otherwise checks the equality with the session name used for the test.
interactionType	Checks the equality with the expected interaction type.
interactionStage	Checks the equality with the expected interaction stage.
transactionId	Checks that the transaction identifier is the same for all the messages returned by the provider.
area	Checks the equality with the expected area.
service	Checks the equality with the expected service.
operation	Checks the equality with the expected operation.
version	Checks the equality with the expected service version.
isError	Checks that the message body is compliant with the value of this header.

- catch the MAL errors. If such an error happens, the test fails.
- check the results ordering during the interactions Invoke and Progress if the QoS is not Best Effort.

Finally, after having initiated all the interactions with every possible header values and every possible transitions, the consumer shall call the operation 'getBadHeaderReports' and check that it is empty.

4.1.2 Pub/Sub header

This test checks that the message header is correct during the Pub/Sub interaction. The checking rules are the same as for the other interaction patterns (see 4.1.1) except that the provider is replaced by a broker.

The test scenario 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 the following subscription:

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

The consumer checks the header of the register acknowledgement.

The consumer triggers publications with the entity key “A”. Four publications are done, one for each update type: creation, update, modification and deletion.

The consumer checks:

- the Notify messages arrival
- the Notify message headers correctness

If the QoS is Assured, Queued or Timely, the consumer also checks the Notify messages order thanks to the incremented counter of the TestUpdate structure.

The consumer triggers a publication of an error and checks:

- the Notify message arrival
- the Notify message header correctness (especially the field 'isError').

4.1.3 Pub/Sub interaction

This test verifies more complex aspects of the Pub/Sub interaction.

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

The consumer creates the subscription “sub1” defined in section 4.1.2 with the QoS level Assured and the session LIVE.

The consumer triggers one update publication with the entity key “A”, the update type Update and the QoS level Queued.

The consumer checks:

- the Notify message arrival
- the QoS level is Assured (same as in the subscription)

The consumer keeps the 'transactionId' of the Notify message.

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:

- that each Update arrives only once despite the two entity requests own the same expression
- that the update which type is Update is not received ('only on change' is true)
- that the 'transactionId' is the same as before the subscription redefinition.

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

The consumer triggers an error publication and checks that the error is received by both subscriptions

“sub1” and “sub2”.

4.1.4 Pub/Sub name space

This test verifies that the subscription name space is defined by the domain, network zone and session used by the consumer.

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

The consumer creates two subscriptions in two different domains “Test.Domain1” and “Test.Domain2” as defined below:

subscription id	“sub1”	“sub1”
authenticationId	{0x00, 0x01}	{0x00, 0x01}
qos	Assured	Assured
priority	1	1
domain	*{“Test”, “Domain1”}*	*{“Test”, “Domain2”}*
networkZone	“TestNetwork”	“TestNetwork”
session	Live	Live
session name	“LIVE”	“LIVE”
entity expression	“A”	“A”
only on change	false	false

The consumer triggers 4 publications in this order:

1. One update publication with the entity key “A” and the update type Modification in the domain “Test.Domain1”
2. The same but in the domain “Test.Domain2”
3. An error publication in the domain “Test.Domain1”
4. An error publication in the domain “Test.Domain2”

The consumer checks:

1. the arrival of the Update published from the domain “Test.Domain1” (resp. “Test.Domain2”) to the subscription created in the domain “Test.Domain1” (resp. “Test.Domain2”)
2. the arrival of the Error published from the domain “Test.Domain1” (resp. “Test.Domain2”) to the subscription created in the domain “Test.Domain1” (resp. “Test.Domain2”)
3. the non-arrival of the Update published from the domain “Test.Domain1” (resp. “Test.Domain2”) to the subscription created in the domain “Test.Domain2” (resp. “Test.Domain1”)

The same test shall be done also with two subscriptions having different network zones, session types or session names.

4.1.5 Pub/Sub 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 the following keys are received and only them:

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

4.1.6 Pub/Sub unknown entity

This test checks that the error UNKNOWN is raised if a registration contains unknown entities.

The consumer doesn't call the operation 'addPublishedEntities' and directly registers to an entity “A”. The error UNKNOWN shall be raised by the broker.

4.2 DataTest

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 given to 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 ErrorTest

The following tests use the ErrorTest service.

4.3.1 Destination unknown

A consumer is created with the URI of the provider which is statically known. The provider is started and closed. So its URI is valid but it is not active so it is not possible to contact it.

The consumer calls the operation 'test' with the QoS level Assured and shall receive the error DESTINATION_UNKNOWN.

4.3.2 Destination transient

A consumer is created with an inactive URI. The consumer calls the operation 'test' and shall receive the error DESTINATION_TRANSIENT.

4.3.3 Delivery timed-out

The provider is started and closed. So its URI is valid but it is not active so it is not possible to contact it. The consumer calls the provider with the QoS Queued and with a TTL. The IP initiation message will be queued and will expire raising the error DELIVERY_TIMEDOUT.

4.3.4 Security

A test security module is to be implemented in order to raise the AUTHENTICATION_FAIL error (resp. the AUTHORIZATION_FAIL error). Plug the test security module into the provider's MAL.

The consumer calls the operation 'test' and checks that the security error is raised.

4.3.5 Encryption

A data structure is modified in order to raise the error ENCRYPTION_FAIL when it is encoded. The consumer calls the operation 'testEncryption' and catches the error ENCRYPTION_FAIL raised locally.

A data structure class is not declared on the provider side so that the decryption fails. The consumer calls the operation 'testEncryption' and catches the error BAD_ENCODING.

4.3.6 Area, operation, version

A service provider is to be implemented for the test in order to raise those errors. Three operations are provided:

- raiseAreaError
- raiseOperationError
- raiseVersionError

4.4 PriorityTest

The consumer asynchronously calls N times the operation 'invokeAction' provided by the service PriorityTest with a low priority level.

Then it synchronously calls one more time the operation 'invokeAction' with a high priority level and checks that the returned counter value is less than N-1. This shows that the high level message has been handled before the last low level message.

Remark:

> The 'sleepDelay' has to be long enough to enable the consumer to make the high priority call before the last low priority message is handled by the provider.

5 Data structures

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

5.1 IPTest structures

5.1.1 IPTestDefinition

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

Structure Name	IPTestDefinition	
Extends	MAL::Composite	
Abstract		
Field	Type	Comment
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

5.1.2 SendTestDefinition

This data structure just make the IPTestDefinition concrete. Nothing more is defined as the Send interaction is one way: the provider doesn't return any message to the consumer.

Structure Name	SendTestDefinition
Extends	IPTestDefinition
Short form	test_ip_setd

Field	Type	Comment

5.1.3 SubmitTestDefinition

This data structure specifies how the IPTest provider shall handle a Submit interaction.

Structure Name	SubmitTestDefinition	
Extends	IPTestDefinition	
Short form	test_ip_sutd	
Field	Type	Comment
ack	MAL::Boolean	If true, the test requires that the provider returns an acknowledgement, otherwise the error TEST_ERROR must be returned.

5.1.4 RequestTestDefinition

This data structure specifies how the IPTest provider shall handle a Request interaction.

Structure Name	RequestTestDefinition	
Extends	IPTestDefinition	
Short form	test_ip_rtd	
Field	Type	Comment
res	MAL::Boolean	If true, the test requires that the provider returns a response “RES”, otherwise the error TEST_ERROR must be returned.

5.1.5 InvokeTestDefinition

This data structure specifies how the IPTest provider shall handle an Invoke interaction.

Structure Name	InvokeTestDefinition	
Extends	IPTestDefinition	
Short form	test_ip_itd	
Field	Type	Comment
ack	MAL::Boolean	If true, the test requires that the provider returns a response “ACK”, otherwise the error TEST_ERROR must be returned (and the test ends).

res	MAL::Boolean	If true, the test requires that the provider returns a response “RES”, otherwise the error TEST_ERROR must be returned.

5.1.6 ProgressTestDefinition

This data structure specifies how the IPTest provider shall handle a Progress interaction.

Structure Name	ProgressTestDefinition	
Extends	IPTestDefinition	
Short form	test_ip_ptd	
Field	Type	Comment
ack	MAL::Boolean	If true, the test requires that the provider returns a response “ACK”, otherwise the error TEST_ERROR must be returned (and the test ends).
stepNumber	MAL::Integer	Number of progress steps expected by the consumer. Each progress update shall contain an integer counter incremented from 0 (first progress update).
res	MAL::Boolean	If true, the test requires that the provider returns a response “RES”, otherwise the error TEST_ERROR must be returned.

5.1.7 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

5.1.8 BadHeaderReportList

This data structure is a list of BadHeaderReport.

List Name	BadHeaderReportList
Short form	test_ip_bhrl
List of	BadHeaderReport

5.1.9 TestPublication

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

Structure Name	TestPublication	
Extends	MAL::Composite	
Abstract		
Field	Type	Comment
update	MAL::Element	The update to be published by the provider
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.

5.1.10 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::Element	The update to be published by the provider

5.1.11 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.

5.1.12 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 ErrorTest structures

5.2.1 ErrorTestData

This data structure is used to produce encryption errors (ENCRYPTION_FAIL and BAD_ENCODING).

Structure Name	TestErrorData	
Extends	MAL::Composite	
Short form	test_err_ted	
Field	Type	Comment
content	MAL::String	A content used to make the encoding failed in the encryption error test.

6 Software development tasks

The development tasks are presented below:

Module	Cost	By who	By when
IP test 1. All patterns except Pub/Sub 2. Pub/Sub header 3. Pub/Sub interaction 4. Pub/Sub name space 5. Pub/Sub entity requests 6. Pub/Sub unknown entity	1 mw	?	?
DataTest	1 md	?	?
Error tests	3 md	?	?
Priority tests	1 md	?	?

Prerequisite: the development tasks require that the XML service definition language is defined and a stub generator implemented.