

SPEC-ION

Last changed:	03.07.2025
Version:	3.0.0-rc.6
Creator:	VDV-ETS

Table of Contents

1	Introduction and Overview	7
1.1	ION Topology	7
1.2	Direct Connection to the ION	8
1.3	Adapter-based Connection to the ION	9
1.4	Joint Service Broker Connection to the ION	10
1.5	Distributed Initiator System	11
1.6	ION Services	13
2	ION Actors	14
2.1	Initiator	14
2.2	Processor	14
2.3	Central routing engine	14
3	Workflows	15
3.1	Process in a synchronous context	15
3.1.1	ION message exchange in a synchronous context	15
3.2	Process in an asynchronous context	16
3.2.1	ION message exchange in an asynchronous context	17
3.3	Supporting activities	18
3.3.1	Compile and send ION request	18
3.3.2	Receive and validate ION request	18
4	Message Types and Scenarios	18
4.1	Get Scenario	19
4.2	Notification Scenario	19
4.3	Mixed Scenario	20
4.4	List and Multiprocessing Scenario	21
4.5	Monitoring and Warning Scenario	22
4.6	Message Types	23
4.6.1	Test Environment Message Types	24
4.6.2	Asynchronous Reply Overview	35
4.6.3	ION message with WSS	36
4.6.4	ION reply message	38
4.6.5	ION request message	38
4.6.6	ION synchronous response message	40
4.6.7	ION synchronous exception message	42
4.6.8	ION asynchronous response message	44
4.6.9	ION asynchronous exception message	46
4.6.10	ION message payload	48
4.6.11	ION request payload	49
4.6.12	ION exception payload	50
4.6.13	Temporarily storable ION message	51
4.6.14	Delivery Acknowledgement	52
4.6.15	Delivery Rejection	52
4.6.16	SOAP fault	53

5	System Components and Interfaces	54
5.1	ION Public Key Infrastructure	55
5.1.1	Certificate Retrieval Service	55
5.2	Central routing engine	55
5.2.1	Central routing engine conceptual interface	55
5.3	Initiator system	56
5.3.1	Initiator interface	56
5.4	Processor system	56
5.4.1	Processor interface	56
6	Use Cases.....	57
6.1	ION Messaging Use Cases	57
6.1.1	Overview ION Messaging Use Cases	57
6.1.2	Synchronous ION use case	61
6.1.3	Asynchronous ION use case	102
6.1.4	Compile and send ION request	147
6.1.5	Supporting activities.....	152
6.1.6	Supporting actions	167
6.2	CRE Use Cases	176
6.2.1	Overview of the central routing engine use cases	176
6.2.2	Service availability notification	178
6.2.3	Conceptual routing	185
6.2.4	Store & Forward	192
6.2.5	Monitoring and notification	206
6.2.6	Supporting activities.....	210
6.2.7	Supporting objects	214
6.2.8	Supporting actions	216
6.3	Supporting objects	219
6.3.1	Keys and certificates	219
6.3.2	Message parts and instances	223
6.3.3	Configurations.....	227
7	Non-Functional Requirements	229
7.1	Process Instance ID Handling	229
7.1.1	Handle process instance ID	229
7.2	Message Timeout and Retry Handling	231
7.2.1	Timeout Handling.....	232
7.2.2	Retry Handling	234
7.3	Version Management.....	239
7.3.1	One Version per Participant.....	239
7.3.2	One Participant serves two Versions.....	240
7.3.3	Each Participant serves two Versions	241
7.4	Common Non-Functional Requirements	241
7.4.1	Messaging requirements	241
7.4.2	Performance.....	249
7.4.3	Scalability	250
7.5	Webservice Security	251



7.5.1	Key Management	251
7.5.2	Message-based encryption (application layer)	254
7.5.3	Appendix: etiCORE Standard-Policy	258
7.5.4	Appendix: etiCORE Empty-Policy	259
8	etiCORE-Model API	261
8.1	ion	261
8.1.1	BusinessAcknowledgement	261
8.1.2	BusinessException	261
8.1.3	CommunicationInformation	261
8.1.4	CompactCommunicationInformation	262
8.1.5	CompactMessage	262
8.1.6	CompactRequest	263
8.1.7	CompressedBusinessReplyList	263
8.1.8	CompressedXmlList	263
8.1.9	DiscardedMessages	263
8.1.10	DiscardedMessagesMetadata	263
8.1.11	Error	265
8.1.12	Event	265
8.1.13	EventDescription	265
8.1.14	IonMessageId	266
8.1.15	IonMessageNumber	266
8.1.16	Message	266
8.1.17	OperationName	267
8.1.18	OriginalRequestCommunicationInformation	267
8.1.19	ProcessInstanceId	267
8.1.20	RepeatCounter	267
8.1.21	Reply	267
8.1.22	Request	268
8.1.23	Warning	268
8.1.24	WarningList	268
8.2	ion-enums	268
8.2.1	ServiceName	268
8.2.2	ProcessNameEnum	268
8.2.3	ServiceNameEnum	274
8.3	ion-event-enums	275
8.3.1	IonErrorEnum	275
8.3.2	IonWarningEnum	276
8.4	ion-event-types	276
8.4.1	E_ION_DATA_OF_SOAP_HEADER_DOES_NOT_MATCH_MESSAGE_DATA	277
8.4.2	E_ION_DUPLICATE_ION_MESSAGE_ID	277
8.4.3	E_ION_DUPLICATE_RESPONSE	277
8.4.4	E_ION_OPERATION_NOT_IMPLEMENTED	277
8.4.5	E_ION_ORIGINAL_REQUEST_NOT_FOUND	277
8.4.6	E_ION_RECEIVER_ORGANISATION_MISMATCH	277
8.4.7	E_ION_RECEIVER_ROLE_MISMATCH	277
8.4.8	E_ION_RECEIVER_SERVICE_MISMATCH	277

8.4.9	E_ION_SAME_MESSAGE_IN_PROGRESS	277
8.4.10	E_ION_UNKNOWN_SENDER	277
8.4.11	E_ION_UNKNOWN_TECHNICAL_PROBLEM	278
8.4.12	E_ION_WRONG_SENDER_ROLE	278
8.4.13	E_ION_WRONG_SENDER_SERVICE	278
8.5	ion-communication	278
8.5.1	DeliveryAcknowledgement	278
8.5.2	deliveryAcknowledgement	278
8.5.3	DeliveryRejection	278
8.5.4	deliveryRejection	279
8.5.5	InterfaceVersion	279
8.5.6	IonRoutingHeader	279
8.5.7	ionRoutingHeader	281
8.5.8	LengthRestrictedOrganisationId	281
8.5.9	OptionalRoutingInfo	281
8.5.10	DeliveredToEnum	281
8.6	ion-communication-event-enums	281
8.6.1	IonCommunicationErrorCode	281
8.6.2	IonCommunicationErrorEnum	281
8.7	ion-communication-event-types	282
8.7.1	E_IONC_CRE_COULD_NOT_STORE_MESSAGE	282
8.7.2	E_IONC_HEADER_TO_TLS_CERTIFICATE_MISMATCH	282
8.7.3	E_IONC_INVALID_SOAP_HEADER	282
8.7.4	E_IONC_NO_TARGET_ADDRESS_FOUND_FOR_ROUTING_PARAMETERS	283
8.7.5	E_IONC_RECEIVER_NOT_REACHABLE	283
8.7.6	E_IONC_RECEIVER_RESPONSE_TIMEOUT	283
8.7.7	E_IONC_UNKNOWN_OPERATION	283
8.7.8	E_IONC_UNKNOWN_OR_UNAUTHORISED_RECEIVER	283
8.7.9	E_IONC_UNKNOWN_OR_UNAUTHORISED_SENDER	283
8.8	cre-messaging	284
8.8.1	setServiceAvailable	284
8.8.2	SetServiceAvailable	284
8.8.3	setServiceAvailableException	284
8.8.4	setServiceAvailableResponse	284
8.8.5	SetServiceAvailableResponse	284
8.8.6	setServiceUnavailable	284
8.8.7	SetServiceUnavailable	284
8.8.8	setServiceUnavailableException	284
8.8.9	setServiceUnavailableResponse	284
8.8.10	SetServiceUnavailableResponse	284
8.9	cre-event-enums	285
8.9.1	CreErrorEnum	285
8.10	cre-event-types	285
8.10.1	E_CRE_SERVICE_NOT_CONFIGURED	285

Appendix: List of References 286

List of Specification References	286
--	-----



List of Webservice Security References	286
--	-----

1 Introduction and Overview

This specification contains all aspects of the etiCORE ION (Interoperable Network) communication.

Included are:

- Use cases for sending and receiving messages in both synchronous and asynchronous contexts
- Use cases for routing and queuing messages performed by the central routing engine (CRE)
- Use cases of the CRE for participating systems (e.g. toggling the availability of an organisational unit's services)
- Information on message structure
- Information on connection types and technical aspects
- Security aspects for transport layer and message layer

This specification defines the rules concerning the communication of the back-office systems of the public transport companies, as well as the central components such as hotlist service, the central routing engine and the scheme manager systems (application owner, registrar and user medium / SAM management systems).

It contains all aspects of the ION communication specification, including the use cases for sending and receiving messages in both synchronous and asynchronous contexts.

To explain the data exchange between systems, the diagrams often omit the central routing engine, since it is transparent for the communication and simplifies the diagrams. However, each scenario contains diagrams with and without the central routing engine.

For the real communication, the central routing engine is always involved.

The employed protocol is SOAP 1.1 using web services.

The communication inside the ION is secured with three methods:

- TLS transport encryption including client authentication (abbreviation "tls" in the following chapters) to secure the HTTP ("https")
- End-to-end message encryption using the XML encryption standard (abbreviation "enc" and "XML-ENC" in the following chapters), see [\[6\] XML Encryption Standard](#)
- Message signature using the XML signature standard (abbreviation "sig" and "XML-SIG" in the following chapters), see [\[7\] XML Signature Standard](#)

For each method (tls, enc, sig), a private/public key pair is used. Details can be found in the chapter [Webservice Security](#).

1.1 ION Topology

etiCORE uses a network structure with central connections to the ION.

A central structure has the following advantages:

- Centralisation makes it easier to implement future functional extensions, e.g. with regard to monitoring or other protocols
- Centralisation makes it easier to monitor the operation of the participating systems
- Centralisation simplifies communication in such a way that the participating systems only connect to one central system. Especially in the case of asynchronous communication with store & forward in the high available CRE, the participating systems do not have to consider the availability of the other ION participants' systems

- Participating systems only need to allow traffic to and from a single system instead of all other systems (firewall settings, internet security)

The ION itself refers to the entire network that is logically and physically required to send messages throughout Germany.

To set up the ION, the [Scheme Manager](#) of the (((eTicket Deutschland, provides a central routing engine (CRE). Central [Scheme Manager](#) systems that exist only once are, therefore, directly connected to the CRE.

Furthermore, there is a directory service of a PKI which holds the certificates of all keys of the participating organisations for the transport and message encryption, as well as the message signature.

This PKI is operated on behalf of the [Scheme Manager](#).

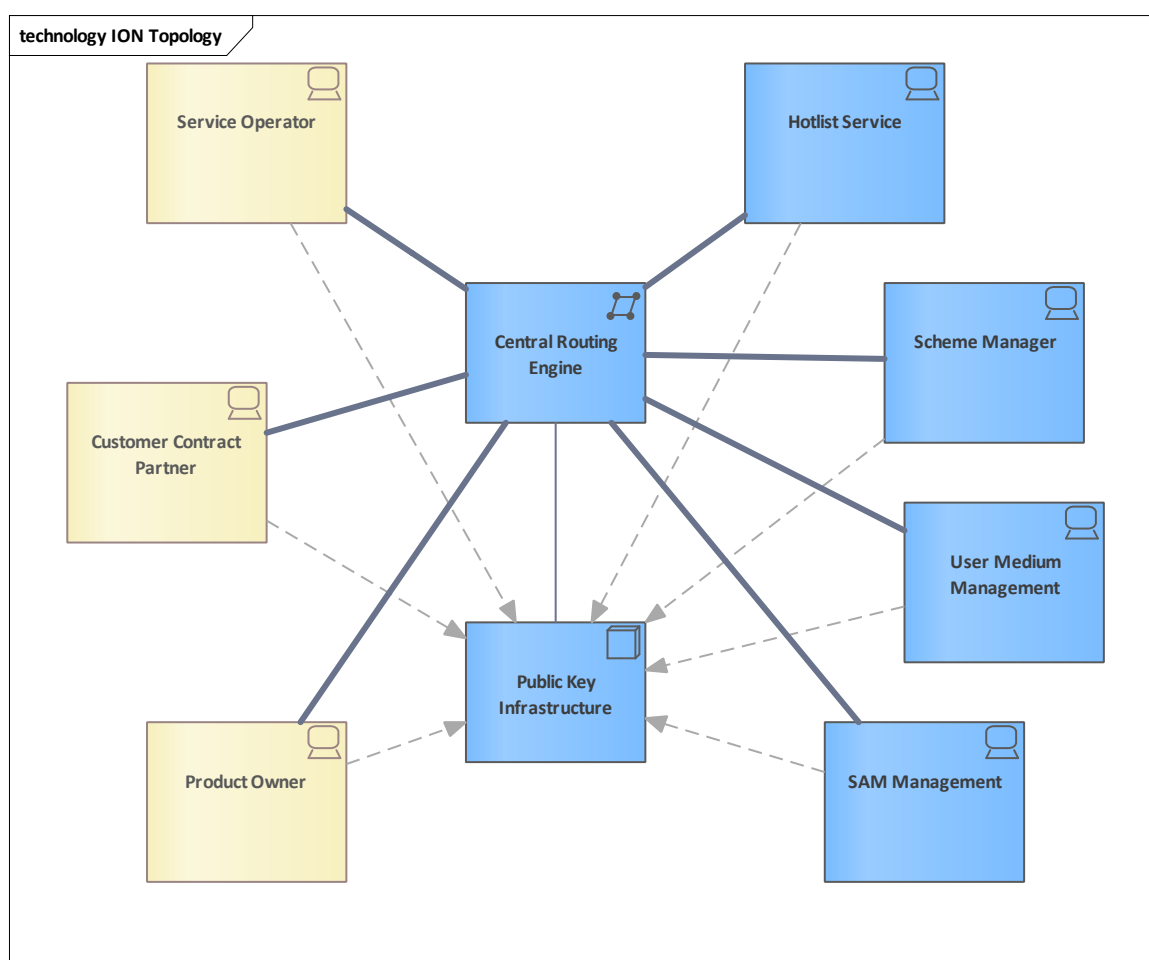


Figure 1: ION Topology

See [ION Topology](#).

1.2 Direct Connection to the ION

With this type of connection, the organisation itself provides the access point to the ION and communicates directly with the CRE and PKI for the secure exchange of ION messages. All parts of the ION security have to be implemented in the participating system.

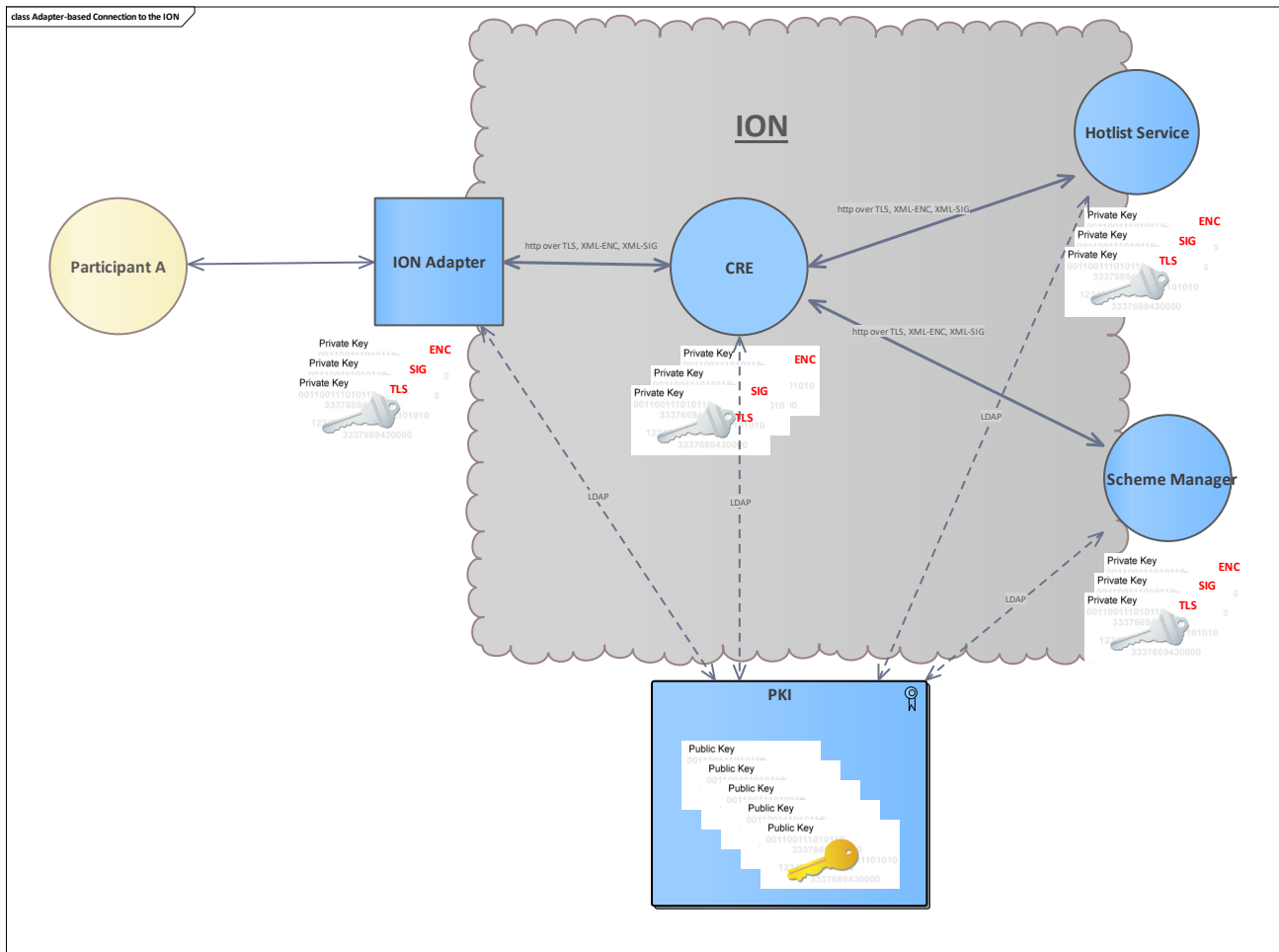


Figure 3: Adapter-based Connection to the ION

See [Adapter-based Connection to the ION](#).

1.4 Joint Service Broker Connection to the ION

In this case, the organisation is connected to a common joint service broker (JSB), which has its own access point to the ION. This requires that the private keys of the participants for message encryption and signature purposes are stored securely at the access point of the JSB, which usually also performs other tasks for participant systems as a common service.

Since only the JSB establishes a TLS connection to the CRE, only a TLS key and certificate are required for it. Participating systems "behind" the JSB do not need their own ION key pair for TLS.

The JSB can be requested and configured in the Scheme Manager system.

The JSB can be built as one system; an implementation with several ION adapters with a simple router could also be possible.

For certain preconditions to employ a connection via JSB, please see [Transport encryption via TLS](#).

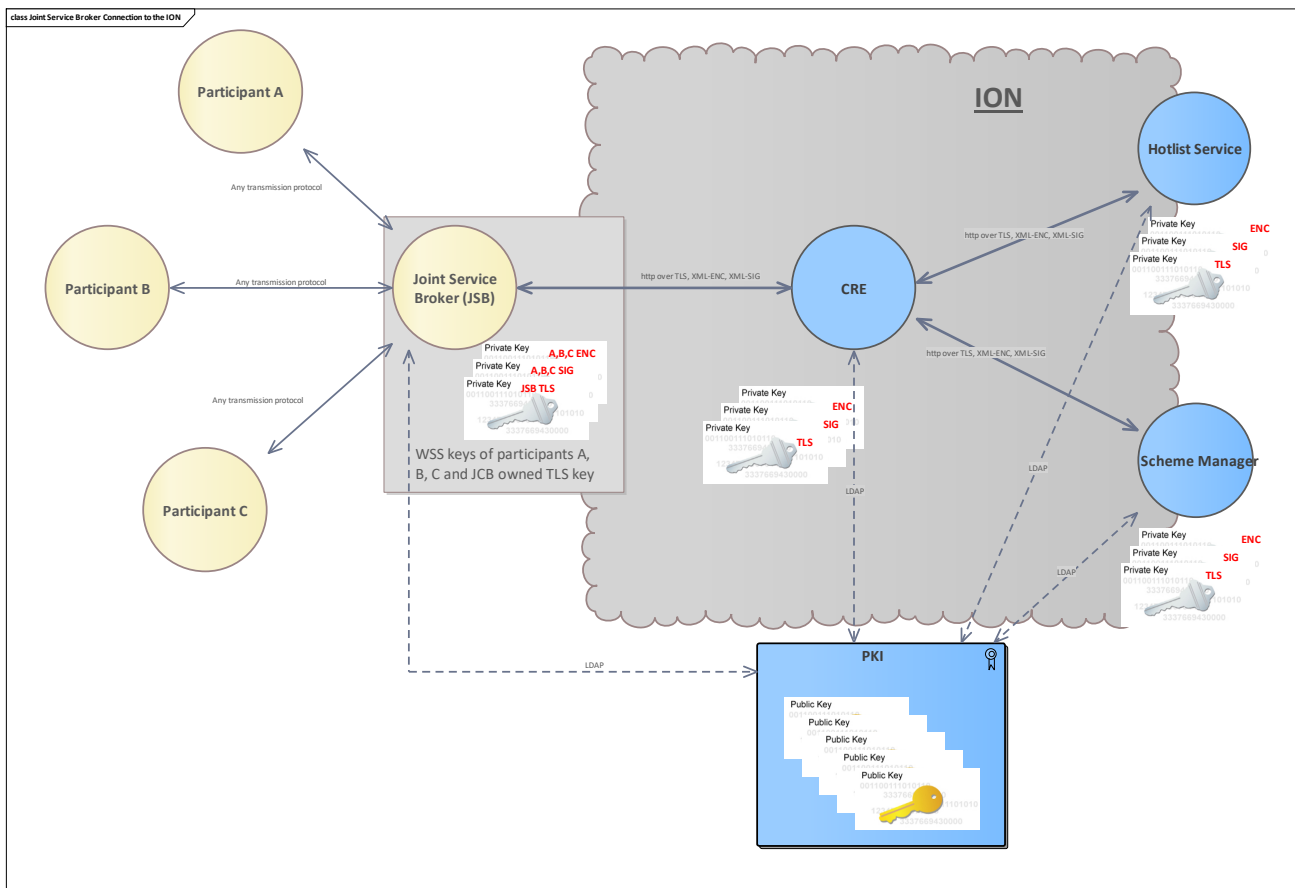


Figure 4: Joint Service Broker Connection to the ION

See [Joint Service Broker Connection to the ION](#).

1.5 Distributed Initiator System

In this constellation, an initiator's distribution component serves as a redirecting service for different client system components.

A certain system of an [Initiator](#) could be split into several components so that one service is distributed to several system components.

Example:

The initiator WSDL of its service provides 6 operations; 3 operations are implemented by the one component and 3 operations by the other. This results in the following challenges:

- In the [Central routing engine](#) only one URL can be configured for one service or WSDL
- Asynchronous use cases raise a problem, since the asynchronous response has to be routed to the right initiator's system component, but this information is missing

For this problem, a new routing option was enabled in the [ionRoutingHeader](#). The field is called *optionalRoutingInfo*.

This field may contain any string with a maximum length of 512 characters. Here, the initiator system can add any information which supports its individual routing purposes. The content can be a system ID that identifies the system component, a URL to this component would also be possible. The ION specification does not impose any requirements for this content.

To support this routing option, the system of the [Processor](#) has to copy this field into the [ionRoutingHeader](#) of its asynchronous response.

Now the distribution component is involved:

- The distribution component must be the TLS endpoint for the whole initiator system. This URL is configured in the CRE
- The distribution component must provide the TLS endpoint and then route or redirect the message due to the information found in *optionalRoutingInfo*
- In this scenario, the distribution component does not employ any message encryption or signature, so there is no need to have the private or public keys for this purpose. Only the private TLS key is needed
- Each system component, however, must have the private or public keys for message encryption and signature

Note: this scenario only works as long as the distributed system is on the initiator's side. If a processor works with distributed system components, the initiator cannot support this, since it has no information about the processor system infrastructure. In this case, the processor has to evaluate the operation name in the [ionRoutingHeader](#) and must decide, based on a mapping table, to which internal component the current message has to be routed.

Note: if a [Joint service broker connection to the ION](#) scenario is used, the redirect service described above could be integrated there.

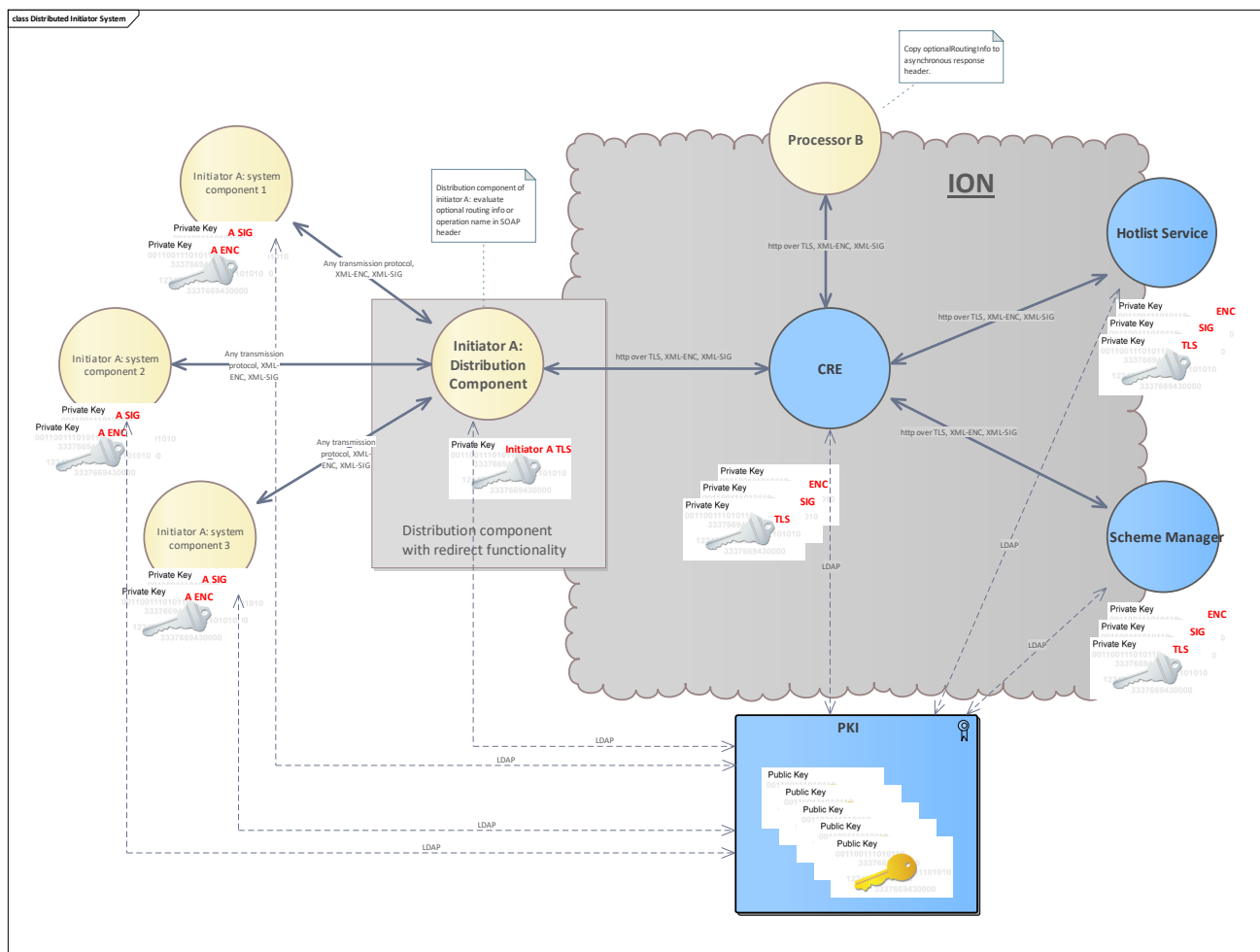


Figure 5: Distributed Initiator System

See [Distributed Initiator System](#).

1.6 ION Services

Overview of the ION services. Each service is represented by its own WSDL. Optional functionality is placed in separate WSDLs and defines further services.

WSDL	Service	Role	Description
ccp.wsdl	ccp	1	Customer contract partner standard service
ccp-oa-execution.wsdl	ccp-oa-execution	1	Optional extension for executing ordered actions by customer contract partner
ccp-oa-ordering.wsdl	ccp-action-ordering	1	Optional extension for the customer contract partner which orders actions
ccp-ste.wsdl	ccp-ste	1	Optional extension for static electronic tickets
ccp-tm.wsdl	ccp-tm	1	Optional extension for tariff modules
ccp-pds.wsdl	ccp-pds	1	Optional extension for product determination service
so.wsdl	so	2	Service operator standard service
so-ste.wsdl	so-ste	2	Optional extension for static electronic tickets
so-tm.wsdl	so-tm	2	Optional extension for tariff modules
po.wsdl	po	3	Product owner standard service
po-oa-management.wsdl	po-oa-management	3	Optional extension for ordered action management and action lists
po-ste.wsdl	po-ste	3	Optional extension for static electronic tickets
po-tm.wsdl	po-tm	3	Optional extension for tariff modules
po-pds.wsdl	po-pds	3	Optional extension for product determination service
asm.wsdl	asm	4	Scheme Manager (including Application and Security Management ASM)
	mms	4	Media Management System
pds.wsdl	pds	4	Product determination service
clearing.wsdl	clearing	4	Service for debt clearing between public transport companies
hotlist.wsdl	hotlist	5	Hotlist standard service
cre.wsdl	cre	254	Central routing engine standard service

2 ION Actors

This package contains the conceptual actors ([Initiator](#) and [Processor](#)) and the real actor [Central routing engine](#) which are used in the ION specification.

2.1 Initiator

Conceptual actor as a placeholder for all actors which initiate a use case that involves message exchanges via the ION.

The initiator is the one that starts a use case. Without sending an initial request, the whole use case would not be performed.

Initially, the initiator sends an [ION request message](#) to the [Processor](#).

In a second step, it receives a message synchronously ([synchronous ION response message](#)) or asynchronously ([asynchronous ION response message](#)).

Note: The initiator always remains the logical initiator, even if - due to the nature of distributed systems and asynchronous processes - the client/server role might change during the runtime of a use case.

2.2 Processor

Conceptual actor as a placeholder for all actors which react to an initial request sent by an [Initiator](#) in a use case that involves message exchanges via the ION.

Depending on the type of the request, it either sends only a [BusinessAcknowledgement](#) response or enhances it with further [Response business data](#).

Note: The processor always remains the logical processor, even if - due to the nature of distributed systems and asynchronous processes - the client/server role might change during the runtime of a use case.

2.3 Central routing engine

Actor which stands for the central routing engine (CRE).

The central routing engine routes messages from an [Initiator](#) to a [Processor](#) and back.

The initiator must pass the routing information, which is defined in the [IonRoutingHeader](#). The CRE uses these parameters stored in this header information to find the configured endpoint of the processor's services:

- Receiver organisation ID
- Receiver role
- Receiver service
- Interface version

The name of the operation is used in the CRE to determine the synchronous or asynchronous context and the timeout behaviour.

In the asynchronous context, the way back from the processor to the initiator works in the same way as described above. If the recipient's system or service is not available, the CRE is able to store the message temporarily and forward it later, when the recipient is available again.

In the synchronous context, the response (or business exception) does not have to be routed, since the open connection is directly used for the response. Thus, no header is needed.

The endpoint URLs of the participating systems can be configured in the CRE via the [Registrar](#) unit of the [Scheme Manager](#) system.

3 Workflows

This package contains the workflow if a synchronous or asynchronous use case takes place. Involved are the [Initiator](#), the [Central routing engine](#) (CRE) and the [Processor](#). The workflow in BPMN shows the different use cases which are passed through in the different systems.

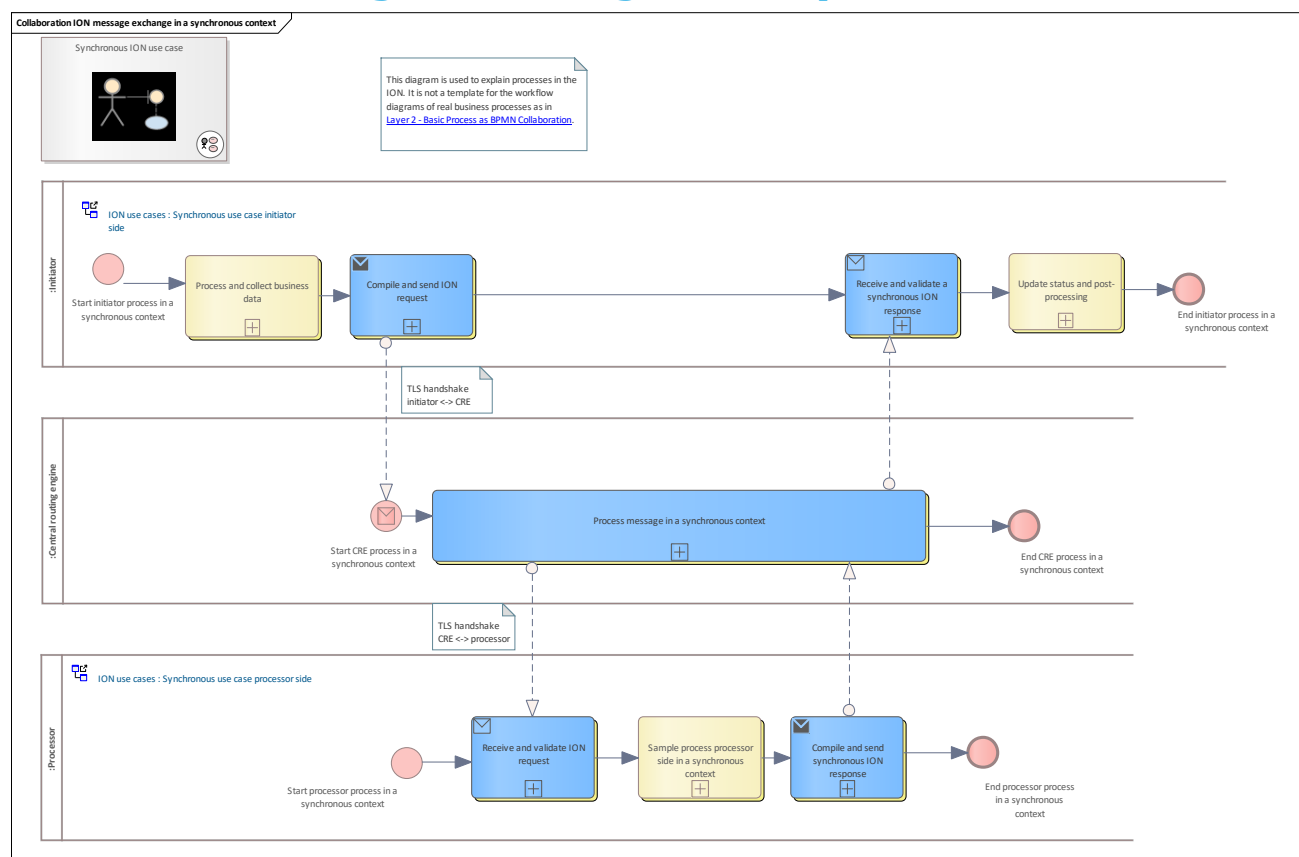
3.1 Process in a synchronous context

Shows the collaboration during a synchronous use case in a BPMN diagram.

All involved parties are shown with their relevant sub-processes.

Note: all use cases are navigable. For the main use case, a hyperlink is placed in the diagram.

3.1.1 ION message exchange in a synchronous context



See [Process in a synchronous context](#).

3.1.1.1 Process and collect business data

See [Sample process initiator side in a synchronous context](#).

3.1.1.2 Receive and validate a synchronous ION response

See [Receive and validate synchronous ION response](#)

3.1.1.3 Update status and post-processing

3.1.1.4 Process message in a synchronous context

See [Process message in a synchronous context](#).

3.1.1.5 Compile and send synchronous ION response

See [Compile and send synchronous ION response](#).

3.1.1.6 Sample process processor side in a synchronous context

See [Sample process processor side in a synchronous context](#).

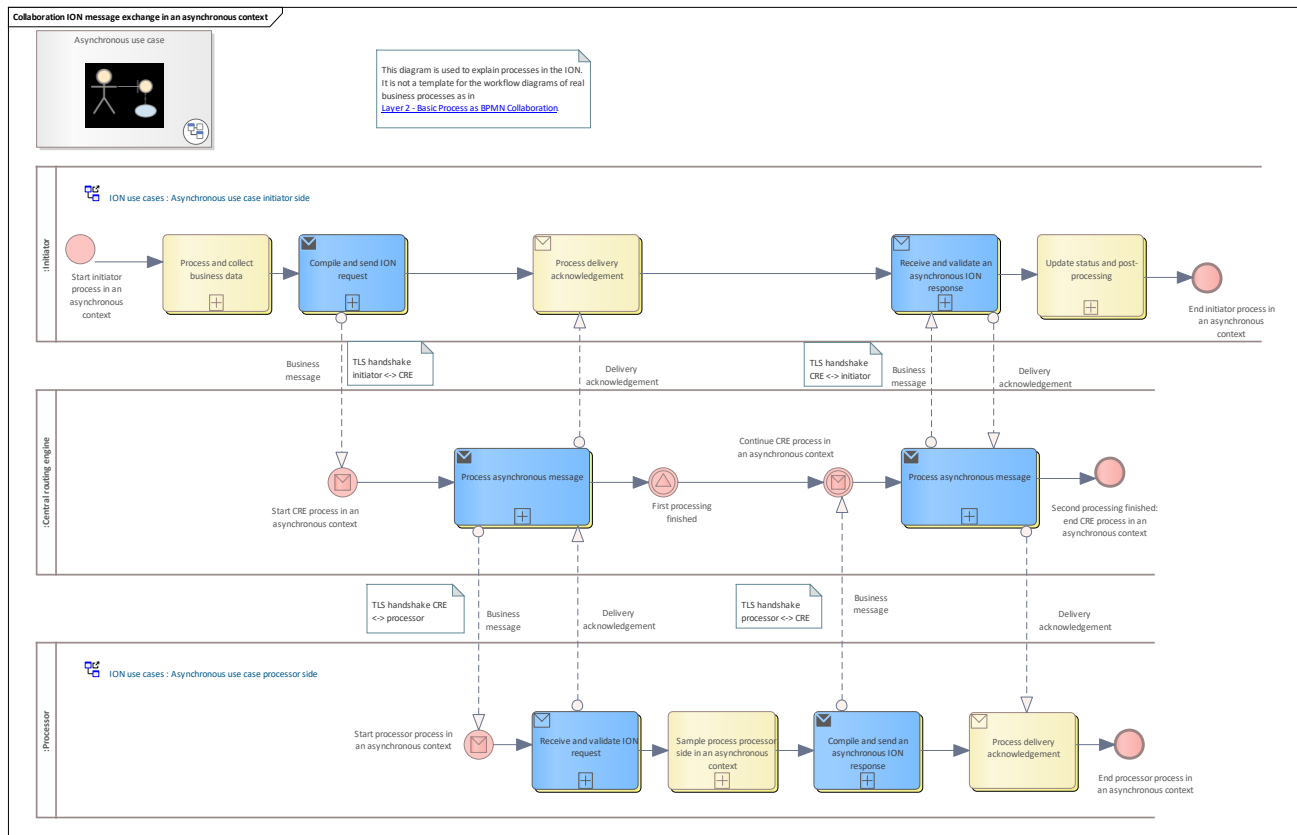
3.2 Process in an asynchronous context

Shows the collaboration during an asynchronous use case in a BPMN diagram.

All involved parties are shown with their relevant sub-processes.

Note: all use cases are navigable. For the main use case, a hyperlink is placed in the diagram.

3.2.1 ION message exchange in an asynchronous context



See [Process in an asynchronous context](#).

3.2.1.1 Process and collect business data

See [Sample process for initiator side in an asynchronous context](#).

3.2.1.2 Process delivery acknowledgement

Registers the (first) delivery acknowledgement. This indicates that the asynchronous use case waits for the business response on the initiator side.

3.2.1.3 Receive and validate an asynchronous ION response

See [Receive and validate asynchronous ION response](#).

3.2.1.4 Update status and post-processing

3.2.1.5 Process asynchronous message

See [Process message in an asynchronous context](#).

3.2.1.6 Process asynchronous message

See [Process message in an asynchronous context](#).

3.2.1.7 Compile and send an asynchronous ION response

See [Compile and send asynchronous ION response](#).

3.2.1.8 Process delivery acknowledgement

Registers the (second) delivery acknowledgement. This indicates that the asynchronous use case is finished since the business response was sent successfully to the initiator.

3.2.1.9 Sample process processor side in an asynchronous context

See [Sample process processor side in an asynchronous context](#).

3.3 Supporting activities

This package contains activities that are used in both synchronous and asynchronous workflows and contexts.

3.3.1 Compile and send ION request

See [Compile and send ION request](#).

3.3.2 Receive and validate ION request

See [Receive and validate ION request](#).

4 Message Types and Scenarios

The following chapter will explain the composition of the ION messages in different scenarios, depending on the underlying use cases (get-scenario, notification-scenario, etc.).

These different scenarios influence the basic contents of the messages.

Thus, all messages belonging to a specific scenario are generally structured in the same way.

The scenarios do not depend on the use case context (see [Synchronous ION use case](#) or [Asynchronous ION use case](#)). The described message compositions inside the scenarios are valid for both contexts (synchronous and asynchronous).

4.1 Get Scenario

In this scenario, an [Initiator](#) system requests data from a [Processor](#) system.

- **Request**

In the simplest case, the request only contains the communication data in the form of a [CommunicationInformation](#), but no business data.

Only the name and type of the request identify the operation to be called in the processor's system to get the requested data.

The request consists of a [request payload](#) with communication information and an optional [Request business data](#) element to transport filters, etc.

In this case, these filters would be the business data for the processor's system.

- **Response**

The response consists of a [response payload](#), containing always a [BusinessAcknowledgement](#) with communication information, a reference to the original request with an optional list of potential events.

Additionally, the response in this scenario must contain data in the form of a [Response business data](#) (e.g. a hotlist, see [getEntitlementHotlistResponse](#)).

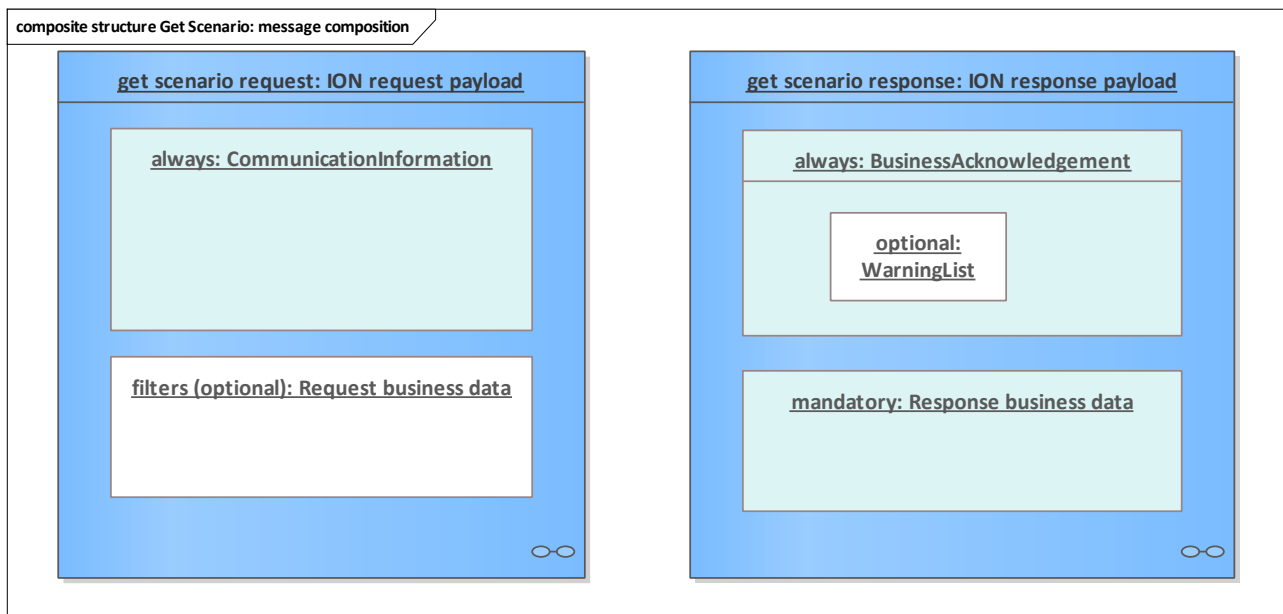


Figure 6: Get Scenario: message composition

4.2 Notification Scenario

In this scenario, the [Initiator](#) system sends a message about a transaction that has already been executed, e.g. [notifyEntitlementIssued](#), as a request to the [Processor](#) system and receives a response about the result of the processing as a business acknowledgement.

- **Request**

A request for a message usually has a complex structure and asks the receiving system to process the reported data. The message of the notification is composed of a [request payload](#) with [CommunicationInformation](#). The [Request business data](#) section of the message contains the data needed to report the event (e.g. the issuance of an entitlement as a TLV encoded binary transaction).

- **Response**

The response to a message is always a [BusinessAcknowledgement](#).

The business acknowledgement does not contain any business data, but signals that processing has taken place successfully.

The business confirmation contains the communication information, a message reference to the original request and, optionally, it can also contain a list of events that can refer to previous messages.

Errors cannot be transported in a business acknowledgement.

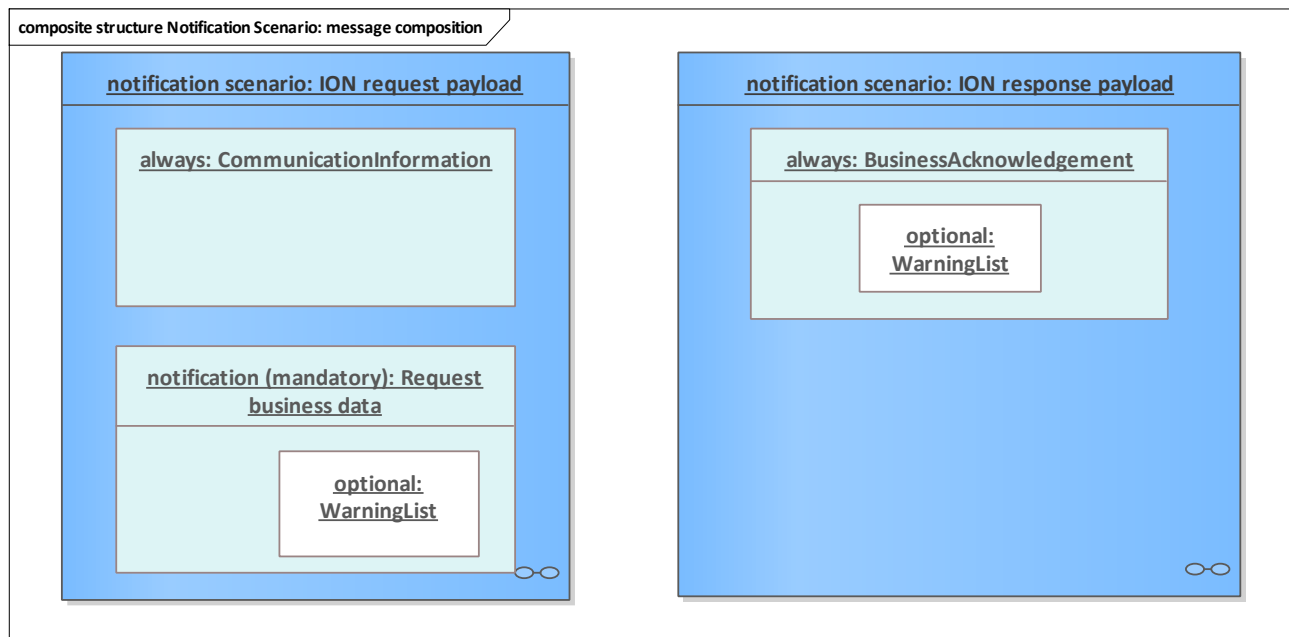


Figure 7: Notification Scenario: message composition

4.3 Mixed Scenario

This scenario covers all other scenarios that cannot be mapped as a [Get Scenario](#) or [Notification Scenario](#). In typical set/add/update/remove operations, the request always has technical data. The response often corresponds to the business confirmation as in the [Notification Scenario](#). In special use cases, both the request and the response have business data.

- **Request**

A mixed scenario request is present if neither a pure data request (get scenario) nor a notification is present. The composition of the request corresponds to the [Notification Scenario](#). The business data area of the request contains the data required for the related use case.

- **Response with business data (for special use cases)**

The business data in the response results from the processing of the business data of the request. The composition of the response often corresponds to the [Get Scenario](#) with business data, but this depends on the use case (for add/remove, for example, the response might not contain additional business data).

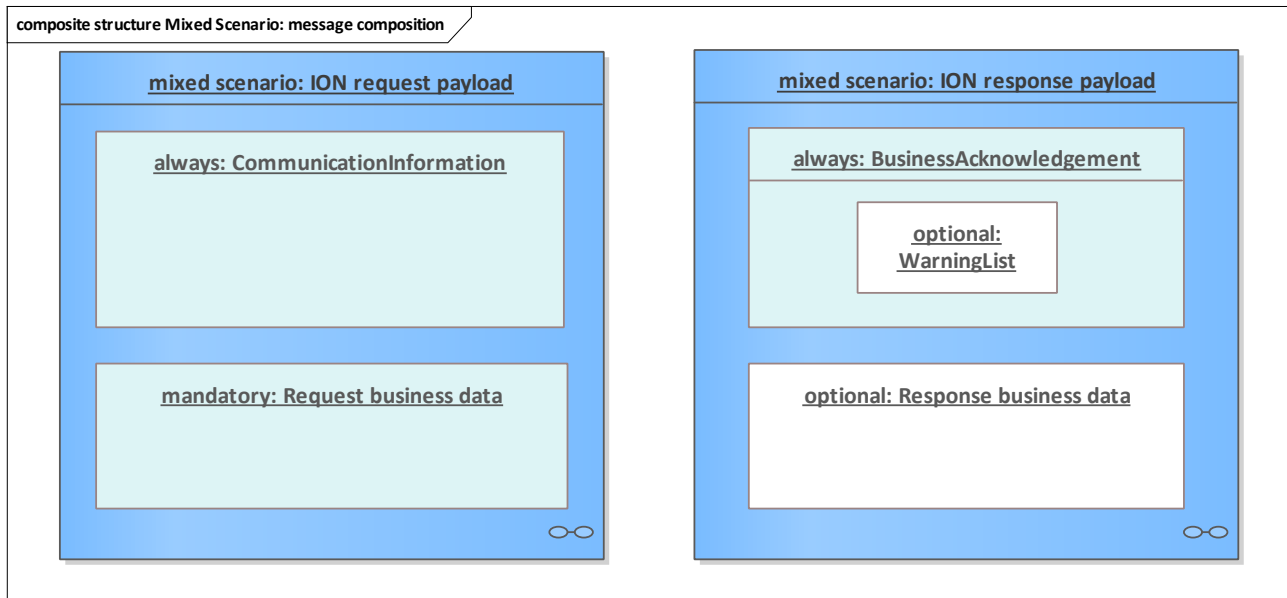


Figure 8: Mixed Scenario: message composition

4.4 List and Multiprocessing Scenario

List scenarios help to transmit messages in a list format and avoid individual calls for each message, especially for larger amounts of data.

The scenario described below can be used independently of synchronous or asynchronous use case context.

In this scenario, a list is transferred for processing during a service call. The confirmation and/or error elements generated in the response are in turn written to a response list.

In etiCORE, list scenarios are only used in connection with notifications. Therefore, multiple uses of the [Notification Scenario](#) are applied.

In interaction with the list scenario, the request elements contain the business data of the message while the response elements only contain confirmations.

If technical errors occur, the response may contain a list of error elements. If some of the messages can be processed and some cannot due to business errors, the response will contain two lists, one with confirmation elements and one with error elements.

In order to prevent redundant data, the usual communication information is not used for the list elements. However, variations that only uniquely identify the list elements, and only together with the communication information of the request or response, contain all communication information of a single message.

Please see [CompactBusinessAcknowledgement](#), [CompactBusinessException](#) (Response) and [IonMessageNumber](#) (Request).

This means that an implemented single processing of a notification can be re-used within the list scenario, but redundant data (e.g. sender ID, sender role, receiver ID, receiver role) is omitted during data transmission. This has to be considered.

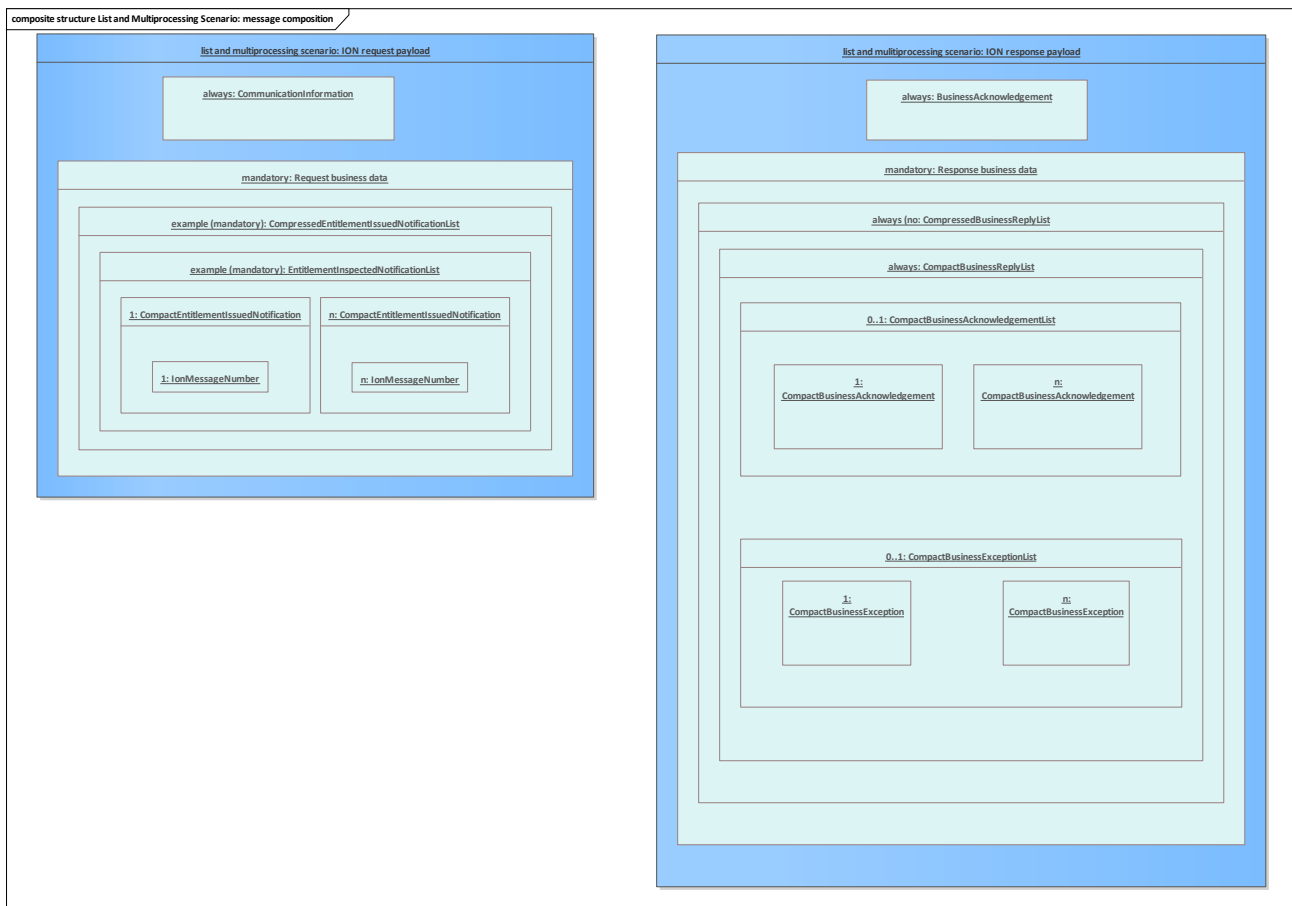


Figure 9: List and Multiprocessing Scenario: message composition

4.5 Monitoring and Warning Scenario

The monitoring and warning scenario occurs when a data situation arises during the downstream processing of data in a background system that is not plausible (e.g. indication of possible fraud by means of forged entitlement).

In this case, a monitoring message is sent from a back-office system to the system that can and must evaluate the situation. The contents of this message are usually warnings with optional references to previous messages.

The monitoring and warning scenario always has a synchronous context. Although manual steps are to be expected in the receiving system, the response is expected to be a business confirmation. If problems occur due to the notification data, this must be clarified by staff.

• Request

The structure of a monitoring message consists of a payload that contains 1..* monitoring events, in addition to the communication information. These, in turn, can refer to 0..* previous messages.

A monitoring event (monitoring check fails) cannot necessarily be directly assigned to a single preceding request.

Rather, a whole series of previously received messages could also have been involved in the creation of the problem. Therefore, each event may relate to one or more previous messages.

Note: The sending system must ensure that no empty monitoring message is sent.

- **Response**

The response to a processing message corresponds to that of the [Notification Scenario](#).

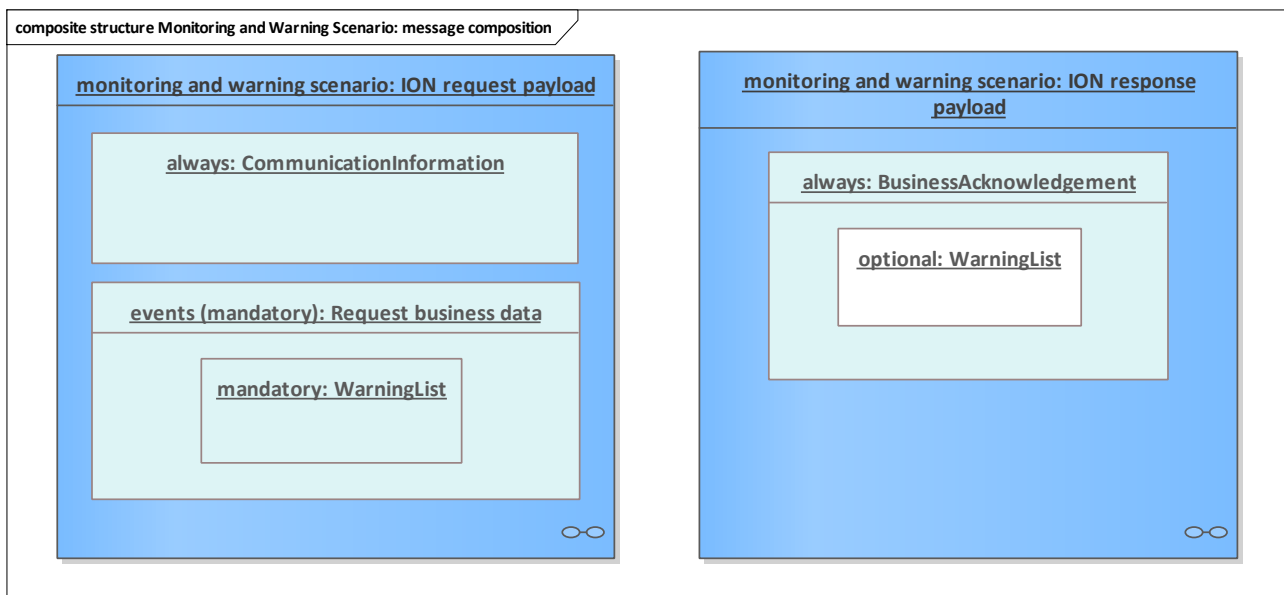


Figure 10: Monitoring and Warning Scenario: message composition

4.6 Message Types

This chapter describes the most relevant ION message types used in the different scenarios.

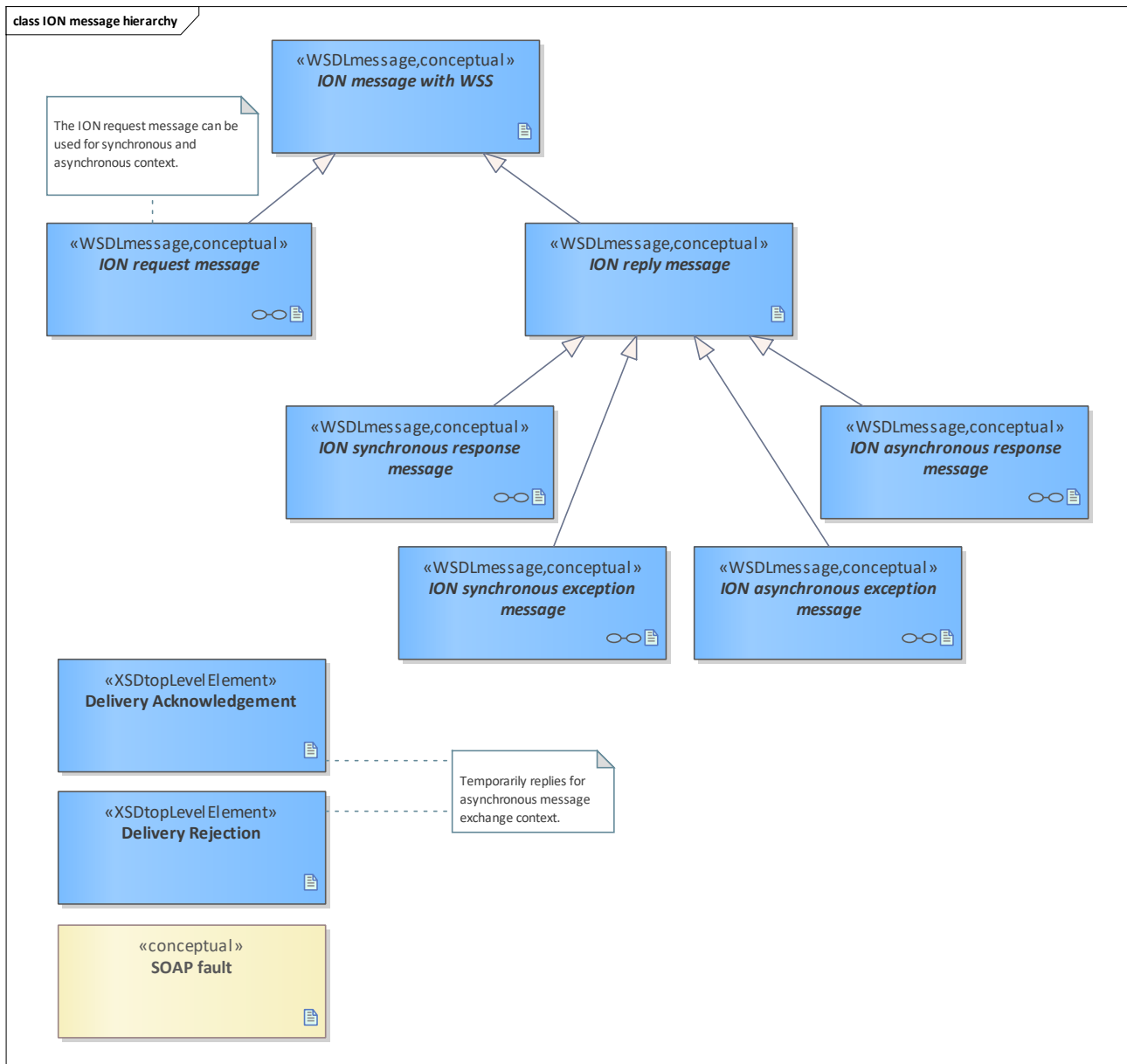


Figure 11: ION message hierarchy

This diagram shows the ION message hierarchy and message types.

Caution: an [ION request message](#) can only be queued if the context is asynchronous. If a synchronous response is expected for the request, the request cannot be queued.

4.6.1 Test Environment Message Types

For test purposes, in a so-called level-2 environment (staging = level-2 environment, production = level-3 environment), messages can be exchanged via the ION without using WSS. This allows functional tests without the complexity of WSS and excludes problems due to incorrect WSS configuration. Additionally, MTOM is switched off, which allows exchanged messages to be logged easily.

The included diagrams show the simplification of the message composition.

For the use cases described later, this means the omission of checks with regard to encryption and signature.

Components are to be implemented in such a way that the system behaviour always respects the embedded WS policy assigned in the binding in the corresponding WSDL. If no WS policy is embedded or assigned, WSS and MTOM are switched off. Considering this, the same system components can work in the test environment as well as the production environment.

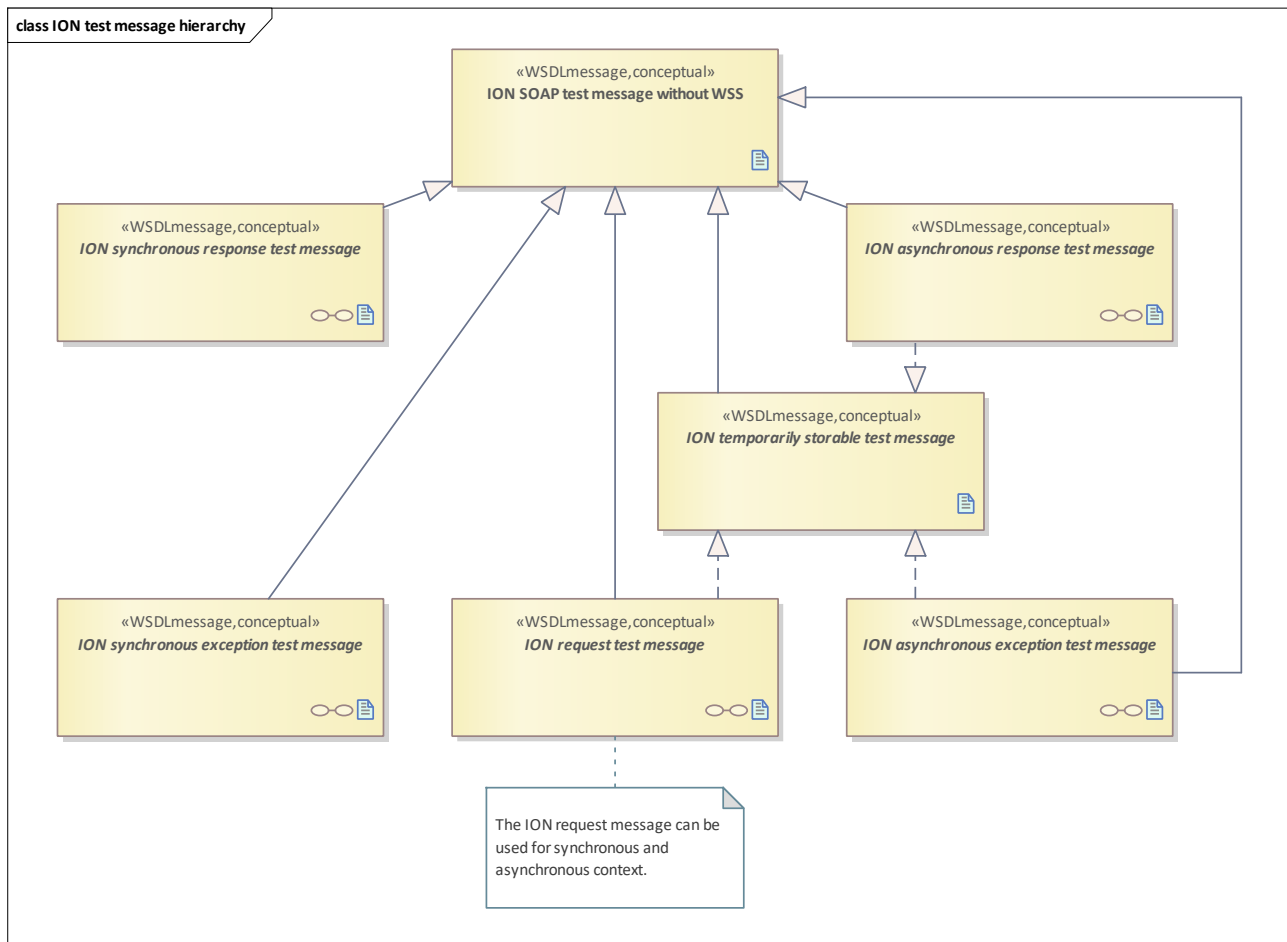


Figure 12: ION test message hierarchy

This diagram shows the test message hierarchy and the test message types which are suitable to be queued inside the CRE for store & forward.

Caution: an [ION request message](#) and its test variant can only be queued if the context is asynchronous. If a synchronous response is expected for the request, the request cannot be queued.

4.6.1.1 ION SOAP test message without WSS

The sample shows an ION SOAP message without security elements for better understanding. Such messages can be employed only in the test environment.

The example uses a message to set a service as being available in the CRE.

In production mode, the ION web service security policy always forces a signature and encryption of the SOAP body due to the etiCORE WSS rules. See also [Webservice Security](#) and [ION SOAP message with WSS](#).

```

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <!-- Not for synchronous responses and business exceptions-->
    <ionRoutingHeader xmlns="https://eticore.org/ion/communication/3">
      <senderId>6598</senderId>
    </ionRoutingHeader>
  </soap:Header>
  <soap:Body>
    <!-- ... -->
  </soap:Body>
</soap:Envelope>
  
```

```
<senderRole>1</senderRole>
<senderService>ccp</senderService>
<messageNumber>4711</messageNumber>
<!--Optional, set only if greater 0:-->
<repeatCounter>1</repeatCounter>
<processInstanceId>SetServiceAvailable_5f7d9b47-b434-416a-9c60-adfde71dec1e</processInstanceId>
<receiverId>5602</receiverId>
<receiverRole>254</receiverRole>
<receiverService>cre</receiverService>
<interfaceVersion>3</interfaceVersion>
<operationName>setServiceAvailable</operationName>
<!--Optional:-->
<optionalRoutingInfo>for regional purposes</optionalRoutingInfo>
</ionRoutingHeader>
</soap:Header>
<soap:Body>
  <cre:setServiceAvailable xmlns="https://eticore.org/ion/3" xmlns:cre="https://eticore.org/cre/messaging/3">
    <communicationInformation>
      <receiverId>5602</receiverId>
      <receiverRole>254</receiverRole>
      <receiverService>cre</receiverService>
      <messageId>
        <senderId>6598</senderId>
        <senderRole>1</senderRole>
        <senderService>ccp</senderService>
        <messageNumber>4711</messageNumber>
      </messageId>
      <messageTimestamp>2022-08-10T08:53:31.327+02:00</messageTimestamp>
      <!--Optional, set only if greater 0:-->
      <repeatCounter>1</repeatCounter>
      <processInstanceId>SetServiceAvailable_5f7d9b47-b434-416a-9c60-adfde71dec1e</processInstanceId>
    </communicationInformation>
  </cre:setServiceAvailable>
</soap:Body>
</soap:Envelope>
```

4.6.1.2 ION request test message

This element defines a conceptual [ION request message](#) as a test version without security elements.

The example for the test message is shown without WSS.

The diagram shows an incoming ION request test message and the related parts.

The diagram shows the composition of the non-encrypted [SOAP Header](#) with its etiCORE header information.

The SOAP body contains the payload in form of a [request payload](#).

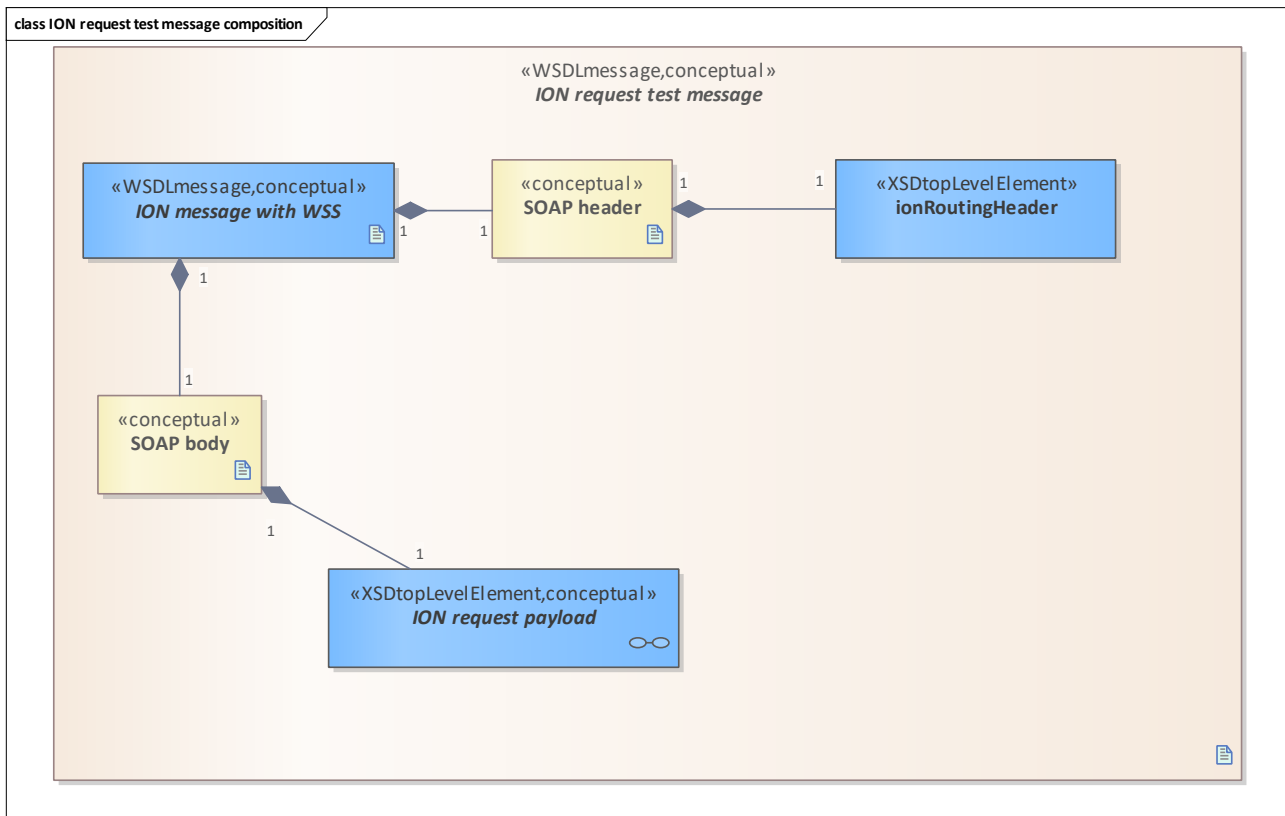


Figure 13: ION request test message composition

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <ionRoutingHeader xmlns="https://eticore.org/ion/communication/3">
      <senderId>6598</senderId>
      <senderRole>1</senderRole>
      <senderService>ccp</senderService>
      <messageNumber>ccp-4710</messageNumber>
      <!--Optional, set only if greater 0:-->
      <repeatCounter>1</repeatCounter>
      <processInstanceId>[To xxx related basic process]_[UUID]</processInstanceId>
      <receiverId>42</receiverId>
      <receiverRole>3</receiverRole>
      <receiverService>po</receiverService>
      <interfaceVersion>3.0.0</interfaceVersion>
      <operationName>xxx</operationName>
      <!--Optional:-->
      <optionalRoutingInfo>for regional purposes</optionalRoutingInfo>
    </ionRoutingHeader>
  </soap:Header>
  <soap:Body>
    <!-- Top-level element xxx instantiates the related data type Xxx
         that extends Request -->
    <po:xxx xmlns="https://eticore.org/ion/3" xmlns:po="https://eticore.org/po/messaging/3">
      <communicationInformation>
        <receiverId>42</receiverId>
        <receiverRole>3</receiverRole>
        <receiverService>po</receiverService>
        <messageId>
          <senderId>6598</senderId>
          <senderRole>1</senderRole>
          <senderService>ccp</senderService>
          <messageNumber>ccp-4710</messageNumber>
        </messageId>
        <messageTimestamp>2022-08-10T08:53:31.569+02:00</messageTimestamp>
        <!--Optional, set only if greater 0:-->
        <repeatCounter>1</repeatCounter>
        <processInstanceId>[To xxx related basic process]_[UUID]</processInstanceId>
      </communicationInformation>
    </po:xxx>
  </soap:Body>
</soap:Envelope>
```

```

<!-- here could be placed additional request business data depending on
the use case and the XSD that defines Xxx -->
</po:xxx>
</soap:Body>
</soap:Envelope>

```

4.6.1.3 ION synchronous response test message

This element defines a conceptual [ION synchronous response message](#) as a test version without security elements.

The example for the test message is shown without WSS.

The diagram shows a synchronous ION response test message derived from an [ION SOAP test message without WSS](#) and the related parts.

The response is characterised by the reference to the corresponding request for this response. Furthermore, an optional list of events is provided in each response.

The message header of the response message differs from the one of a request message, since, due to the synchronicity, routing information is not needed.

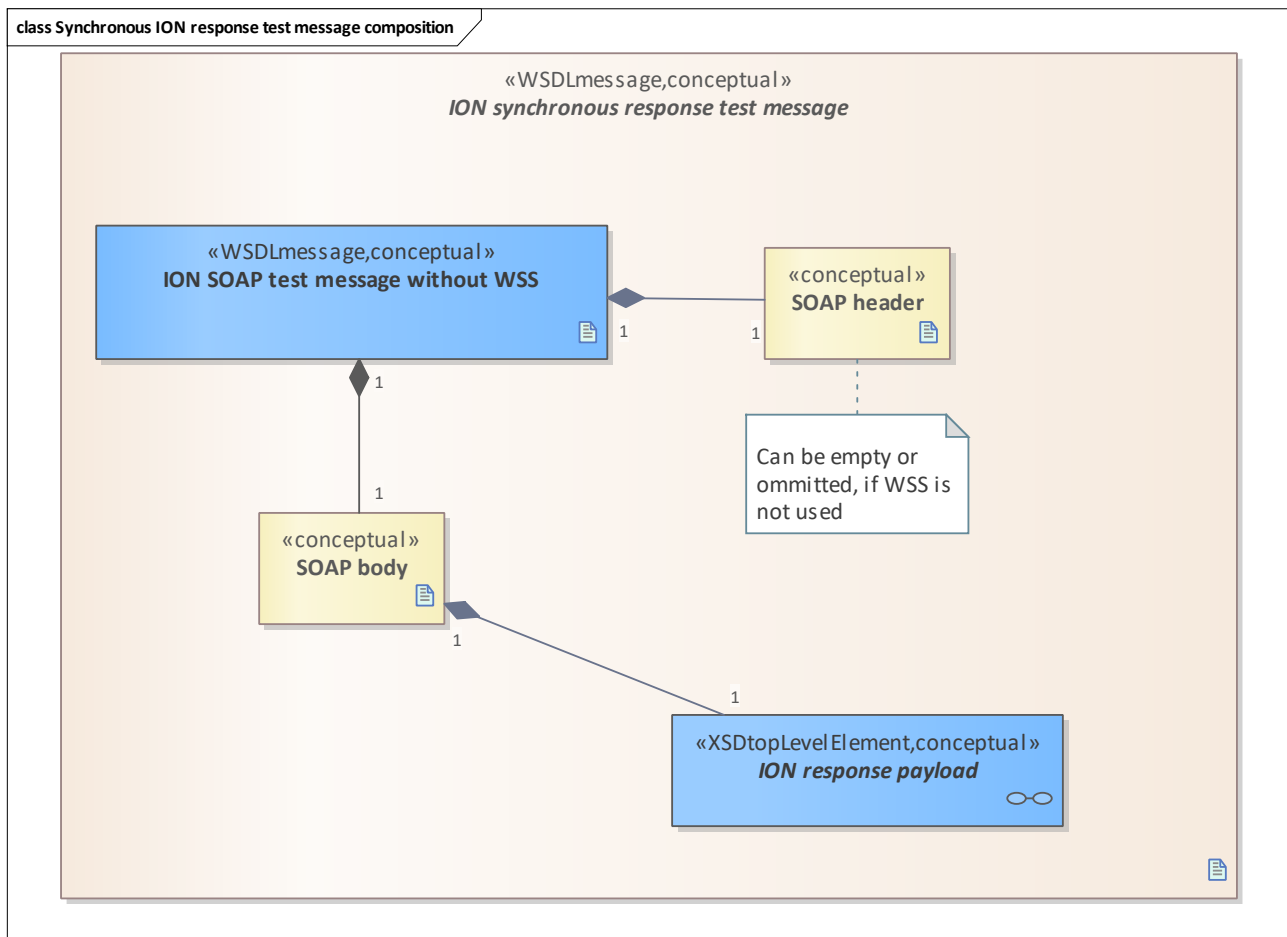


Figure 14: Synchronous ION response test message composition

```

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <!-- no ionRoutingHeader in SOAP header since response is synchronous -->
  <soap:Header>
    <!-- no security header: soap header empty or omit it -->
  </soap:Header>
  <soap:Body>
    <!-- Top-level element xxxResponse instantiates the related data type XxxResponse
that extends BusinessAcknowledgement -->
    <po:xxxResponse xmlns="https://eticore.org/ion/3" xmlns:po="https://eticore.org/po/messaging/3">
      <communicationInformation>

```

```
<receiverId>6398</receiverId>
<receiverRole>1</receiverRole>
<receiverService>ccp</receiverService>
<messageId>
  <senderId>42</senderId>
  <senderRole>3</senderRole>
  <senderService>po</senderService>
  <messageNumber>po-4711</messageNumber>
</messageId>
<messageTimestamp>2022-08-10T08:53:31.569+02:00</messageTimestamp>
<!--Optional, set only if greater 0:-->
<repeatCounter>1</repeatCounter>
<processInstanceId>[To xxx related basic process]_[UUID]</processInstanceId>
</communicationInformation>
<originalRequestCommunicationInformation>
  <originalRequestCorrelationId>
    <senderId>6398</senderId>
    <senderRole>1</senderRole>
    <senderService>ccp</senderService>
    <messageNumber>ccp-4710</messageNumber>
  </originalRequestCorrelationId>
  <originalRequestTimestamp>2022-08-10T08:53:31.327+02:00</originalRequestTimestamp>
  <!--Optional, only if set in request:-->
  <originalRepeatCounter>1</originalRepeatCounter>
</originalRequestCommunicationInformation>
<!-- here could be placed additional response business data depending on
the use case and the XSD which defines XxxResponse -->
</po:xxxResponse>
</soap:Body>
</soap:Envelope>
```

4.6.1.4 ION synchronous exception test message

This element defines a conceptual [ION synchronous exception message](#) as a test version without security elements.

The example for the test message is shown without WSS.

The diagram shows the composition of a ION synchronous exception test message derived from an [ION SOAP test message without WSS](#) and the related parts.

The exception has a reference to the corresponding request for this exception and contains a unique string-based error code. Furthermore, an optional list of additional events is provided. The message header of the response message differs from the one of a request message, since, due to the synchronicity, routing information is not needed.

In the synchronous context, the [exception payload](#) is embedded into a [SOAP Fault](#).

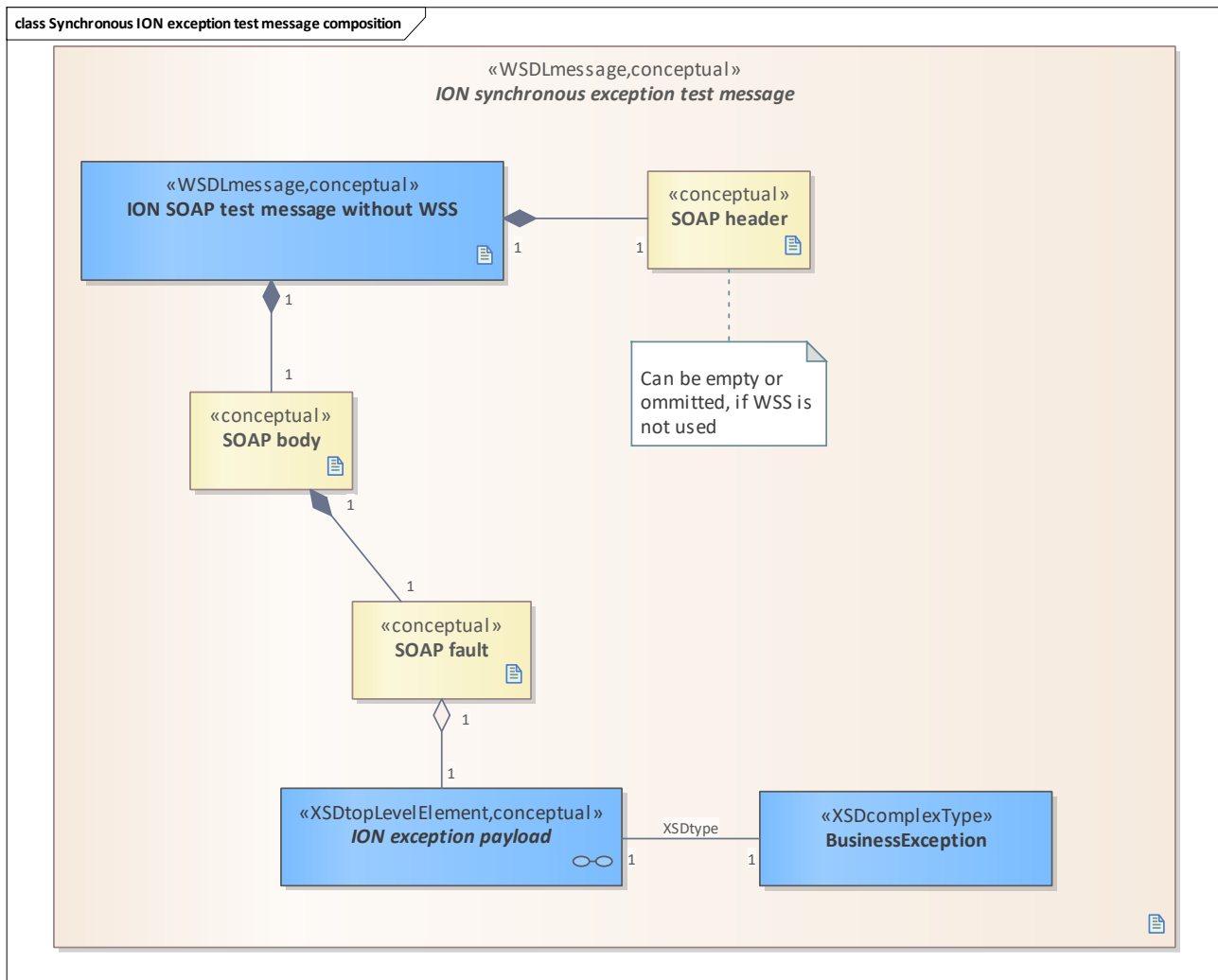


Figure 15: Synchronous ION exception test message composition

```

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <!-- no ionRoutingHeader in SOAP header since response is synchronous -->
  <soap:Header>
    <!-- no security header: soap header empty or omit it -->
  </soap:Header>
  <soap:Body>
    <soap:Fault>

      <!-- Not specified in etiCORE. Field is mandatory
      Recommendation: use the predefined
      term "Server" if the system received the message,
      use the predefined term "Client" if the message
      did not leave the system (e.g. outgoing schema validation failed) -->
      <faultcode>soap:Server</faultcode>

      <!-- If a soap fault is thrown due to security reasons,
      leave the field empty (field is mandatory but nillable).
      For specified exceptions, the string of the detail section
      can be filled here -->
      <faultstring>E_HL_HOTLIST_ENTRY_ALREADY_EXISTS</faultstring>

      <detail>
        <!-- Top-level element addApplicationToHotlistException instantiates
        the data type BusinessException -->
        <hl:addApplicationToHotlistException xmlns:hl="https://eticore.org/hotlist/messaging/3"
        xmlns="https://eticore.org/ion/3">
          <communicationInformation>
            <receiverId>6398</receiverId>
            <receiverRole>1</receiverRole>
          </communicationInformation>
        </hl:addApplicationToHotlistException>
      </detail>
    </soap:Fault>
  </soap:Body>
</soap:Envelope>

```

```
<receiverService>ccp</receiverService>
<messageId>
  <senderId>5600</senderId>
  <senderRole>5</senderRole>
  <senderService>hotlist</senderService>
  <messageNumber>hotlist-4711</messageNumber>
</messageId>
<messageTimestamp>2022-08-10T08:53:31.569+02:00</messageTimestamp>
<!--Optional, set only if greater 0:-->
<repeatCounter>1</repeatCounter>
<processInstanceId>AddApplicationToHotlist_[UUID]</processInstanceId>
</communicationInformation>
<originalRequestCommunicationInformation>
  <originalRequestCorrelationId>
    <senderId>6398</senderId>
    <senderRole>1</senderRole>
    <senderService>ccp</senderService>
    <messageNumber>ccp-4710</messageNumber>
  </originalRequestCorrelationId>
  <originalRequestTimestamp>2022-08-10T08:53:31.327+02:00</originalRequestTimestamp>
  <!--Optional, only if set in request:-->
  <originalRepeatCounter>1</originalRepeatCounter>
</originalRequestCommunicationInformation>
<error>
  <eventIdentifier>E_HL_HOTLIST_ENTRY_ALREADY_EXISTS</eventIdentifier>
  <eventDescription>The hotlist entry already exists in the hotlist service system</eventDescription>
</error>
</hl:addApplicationToHotlistException>
</detail>
</soap:Fault>
</soap:Body>
</soap:Envelope>
```

4.6.1.5 ION asynchronous response test message

This element defines a conceptual [ION asynchronous response message](#) as a test version without security elements.

The example for the test message is shown without WSS.

The diagram shows the composition of a asynchronous ION response test message derived from an [ION SOAP test message without WSS](#) and the related parts.

The diagram shows the [SOAP Header](#) with its etiCORE header information.

The response is characterised by the reference to the corresponding request for this response. Furthermore, an optional list of events is provided in each response.

The message header of the response message does not differ from the one of a request message.

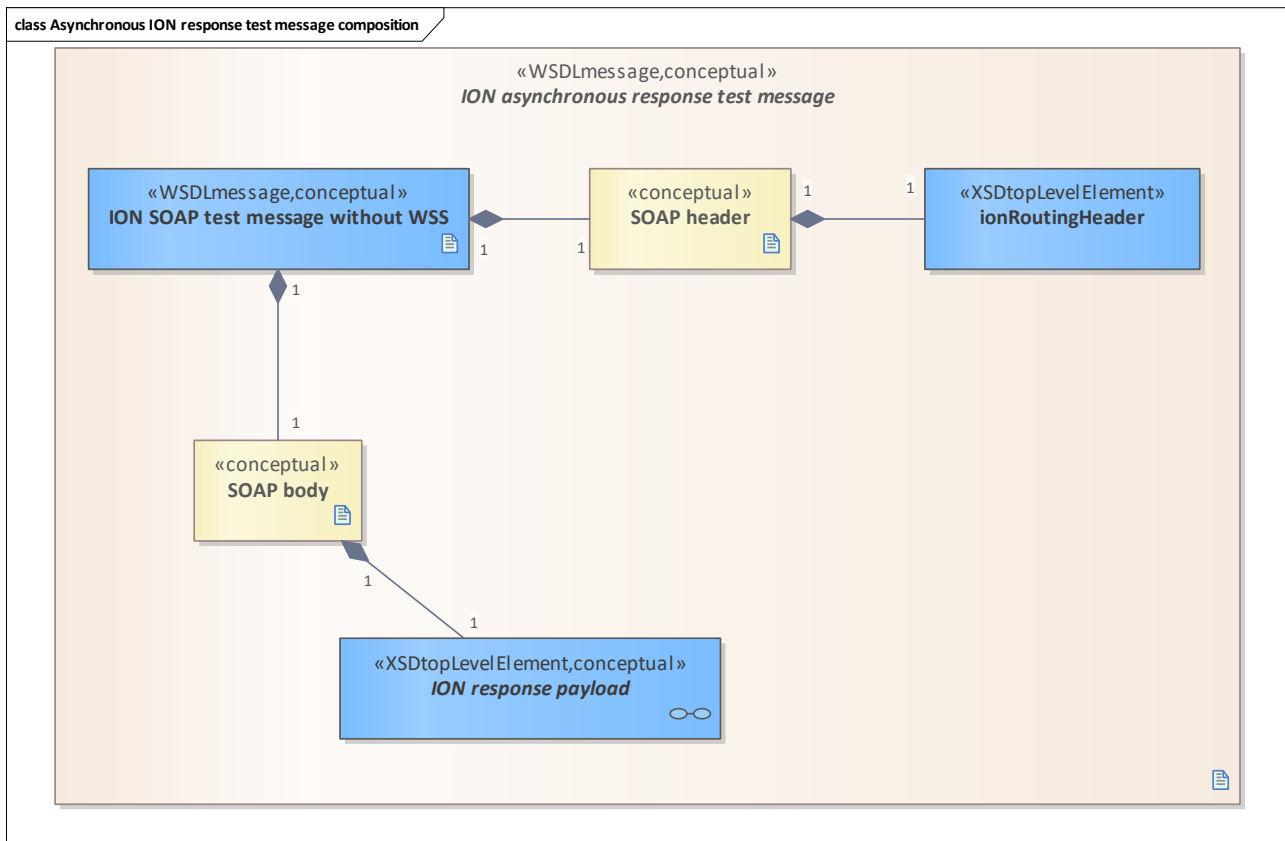


Figure 16: Asynchronous ION response test message composition

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <ionRoutingHeader xmlns="https://eticore.org/ion/communication/3">
      <senderId>42</senderId>
      <senderRole>3</senderRole>
      <senderService>po</senderService>
      <messageNumber>po-4711</messageNumber>
      <!--Optional, set only if greater 0:-->
      <repeatCounter>1</repeatCounter>
      <processInstanceId>[To xxx related basic process]_[UUID]</processInstanceId>
      <receiverId>6598</receiverId>
      <receiverRole>1</receiverRole>
      <receiverService>ccp</receiverService>
      <interfaceVersion>3.0.0</interfaceVersion>
      <operationName>xxxResponse</operationName>
      <!--Optional:-->
      <optionalRoutingInfo>for regional purposes</optionalRoutingInfo>
    </ionRoutingHeader>
  </soap:Header>
  <soap:Body>
    <!-- Top-level element xxxResponse instantiates the related data type XxxResponse
         that extends BusinessAcknowledgement -->
    <ccp:xxxResponse xmlns="https://eticore.org/ion/3" xmlns:ccp="https://eticore.org/ccp/messaging/3">
      <communicationInformation>
        <receiverId>6598</receiverId>
        <receiverRole>1</receiverRole>
        <receiverService>ccp</receiverService>
        <messageId>
          <senderId>42</senderId>
          <senderRole>3</senderRole>
          <senderService>po</senderService>
          <messageNumber>po-4711</messageNumber>
        </messageId>
        <messageTimestamp>2022-08-10T08:53:31.569+02:00</messageTimestamp>
        <!--Optional, set only if greater 0:-->
        <repeatCounter>1</repeatCounter>
      </communicationInformation>
    </ccp:xxxResponse>
  </soap:Body>
</soap:Envelope>
```



```

    <processInstanceId>[To xxx related basic process]_[UUID]</processInstanceId>
  </communicationInformation>
  <originalRequestCommunicationInformation>
    <originalRequestCorrelationId>
      <senderId>6598</senderId>
      <senderRole>1</senderRole>
      <senderService>ccp</senderService>
      <messageNumber>ccp-4710</messageNumber>
    </originalRequestCorrelationId>
    <originalRequestTimestamp>2022-08-10T08:53:31.327+02:00</originalRequestTimestamp>
    <!-- Optional, only if set in request:-->
    <originalRepeatCounter>1</originalRepeatCounter>
  </originalRequestCommunicationInformation>
  <!-- here could be placed additional response business data depending on
       the use case and the XSD which defines XxxResponse -->
</ccp:xxxResponse>
</soap:Body>
</soap:Envelope>

```

4.6.1.6 ION asynchronous exception test message

This element defines a conceptual [ION asynchronous exception message](#) as a test version without security elements.

The example for the test message is shown without WSS.

The diagram shows the composition of a asynchronous ION exception test message derived from an [ION SOAP test message without WSS](#) and the related parts.

The diagram shows the [SOAP Header](#) with its etiCORE header information.

The exception has a reference to the request which caused this exception and contains a unique string-based code. Furthermore, an optional list of additional events is provided.

The message header of the exception message does not differ from the one of a regular request or response.

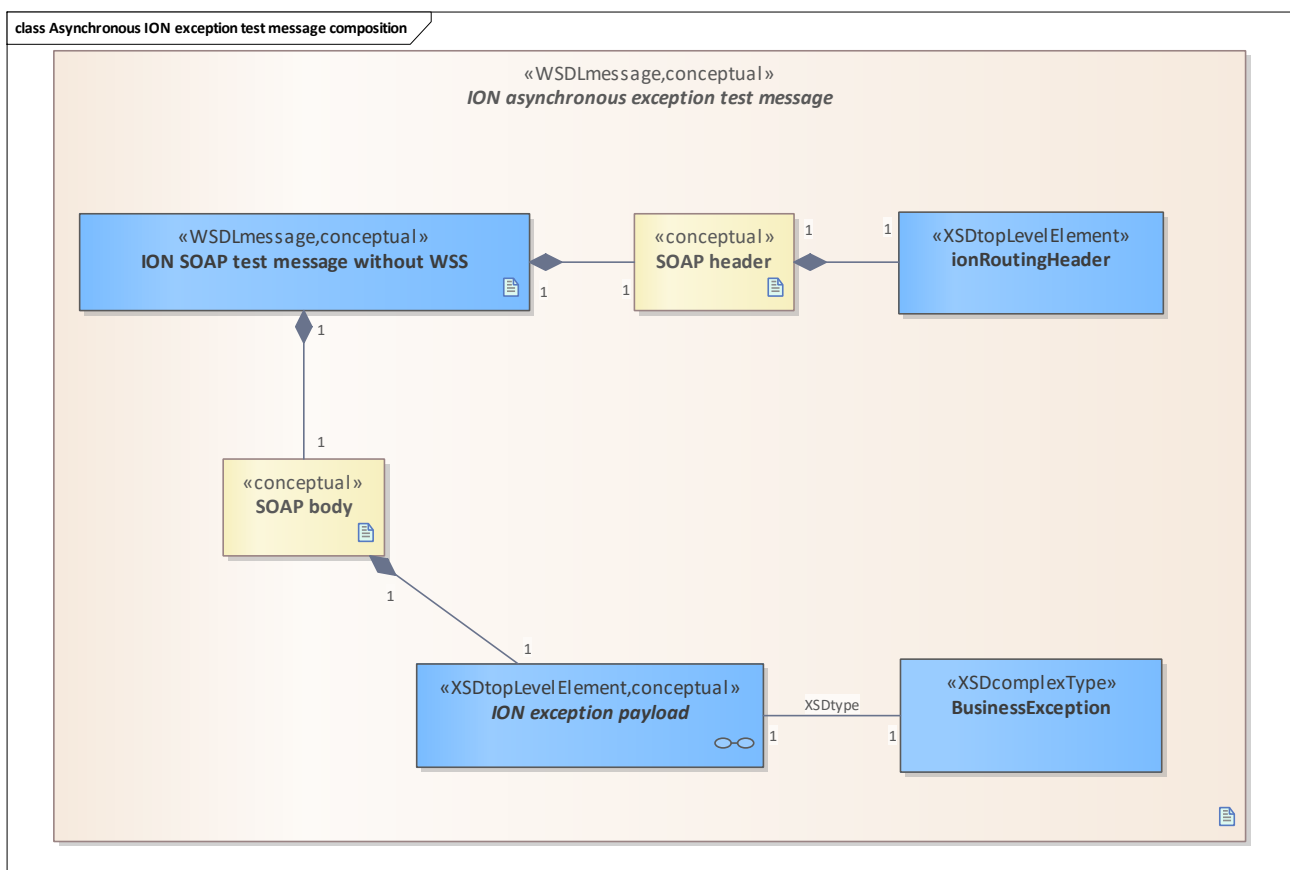


Figure 17: Asynchronous ION exception test message composition

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <ionRoutingHeader xmlns="https://eticore.org/ion/communication/3">
      <senderId>42</senderId>
      <senderRole>3</senderRole>
      <senderService>po</senderService>
      <messageNumber>po-4711</messageNumber>
      <!--Optional, set only if greater 0:-->
      <repeatCounter>1</repeatCounter>
      <processInstanceId>[To xxx related basic process]_[UUID]</processInstanceId>
      <receiverId>6598</receiverId>
      <receiverRole>1</receiverRole>
      <receiverService>ccp</receiverService>
      <interfaceVersion>3.0.0</interfaceVersion>
      <operationName>xxxException</operationName>
      <!--Optional:-->
      <optionalRoutingInfo>for regional purposes</optionalRoutingInfo>
    </ionRoutingHeader>
  </soap:Header>
  <soap:Body>
    <!-- Top-level element xxxException instantiates BusinessException -->
    <ccp:xxxException xmlns="https://eticore.org/ion/3" xmlns:ccp="https://eticore.org/ccp/messaging/3">
      <communicationInformation>
        <receiverId>6598</receiverId>
        <receiverRole>1</receiverRole>
        <receiverService>ccp</receiverService>
        <messageId>
          <senderId>42</senderId>
          <senderRole>3</senderRole>
          <senderService>po</senderService>
          <messageNumber>po-4711</messageNumber>
        </messageId>
        <messageTimestamp>2022-08-10T08:53:31.569+02:00</messageTimestamp>
        <!--Optional, set only if greater 0:-->
        <repeatCounter>1</repeatCounter>
        <processInstanceId>[To xxx related basic process]_[UUID]</processInstanceId>
      </communicationInformation>
      <originalRequestCommunicationInformation>
        <originalRequestCorrelationId>
          <senderId>6598</senderId>
          <senderRole>1</senderRole>
          <senderService>ccp</senderService>
          <messageNumber>ccp-4710</messageNumber>
        </originalRequestCorrelationId>
        <originalRequestTimestamp>2022-08-10T08:53:31.327+02:00</originalRequestTimestamp>
        <!--Optional, only if set in request:-->
        <originalRepeatCounter>1</originalRepeatCounter>
      </originalRequestCommunicationInformation>
      <error>
        <eventIdentifier>E_CO_BINARY_STRUCTURE_CANNOT_BE_PROCESSED</eventIdentifier>
        <eventDescription>The binary structure contained in the request cannot be processed</eventDescription>
      </error>
    </ccp:xxxException>
  </soap:Body>
</soap:Envelope>
```

4.6.1.7 ION temporarily storable test message

Variant for test environment. See [temporarily storable ION message](#) for description that is also valid for test messages.

4.6.1.8 ION response payload

This element represents the conceptual payload part of a response message and instantiates [Response payload type](#).

As a top-level element, it is a placeholder for all business responses sent via the ION. It is embedded in the SOAP body and, due to the etiCORE WSS rules, completely encrypted.

In order to clarify compliance with the recommendations of the WS-I, the name is written in lower case here.

The response payload element is named after the operation which was initially called followed by the suffix "Response", i.e. "<calledOperationName>Response".

For asynchronous responses, the name of the concrete response payload is equal to the name of the operation provided by the [Initiator](#) to receive the response.

The response payload always is a [BusinessAcknowledgement](#) with a [CommunicationInformation](#) object, an [OriginalRequestCommunicationInformation](#) object and an optional (depending on the use case) [Response business data](#) object.

The diagram shows the composition of an abstract response payload including an example.

A concrete response payload always inherits from a [BusinessAcknowledgement](#) and is named after the original operation, followed by the keyword "Response" (e.g. notifyXXResponse).

Furthermore, an optional part of business data may be contained in form of [Response business data](#).

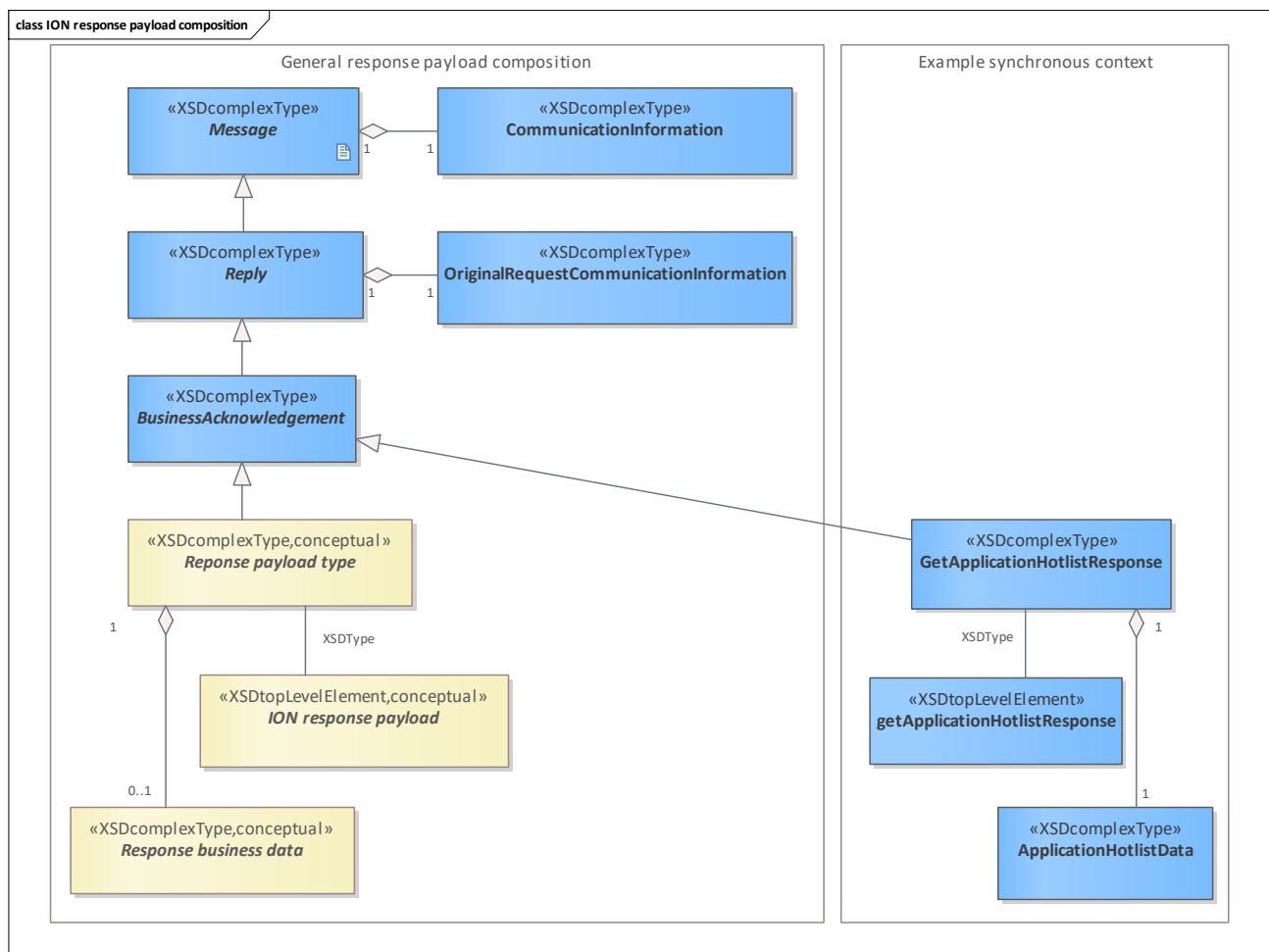


Figure 18: ION response payload composition

4.6.2 Asynchronous Reply Overview

Table that shows the possible (specified) temporarily replies in the asynchronous message exchange context.

Reply	Error Code
deliveryAcknowledgement	-
SOAP Fault with Delivery	E IONC CRE COULD NOT STORE MESSAGE

Reply	Error Code
Rejection	
SOAP Fault with Delivery Rejection	E IONC DATA OF SOAP HEADER DO NOT MATCH MESSAGE DATA
SOAP Fault with Delivery Rejection	E ION DUPLICATE ION MESSAGE ID
SOAP Fault with Delivery Rejection	E ION DUPLICATE RESPONSE
SOAP Fault with Delivery Rejection	E IONC HEADER TO TLS CERTIFICATE MISMATCH
SOAP Fault with Delivery Rejection	E IONC_INVALID_SOAP_HEADER
SOAP Fault with Delivery Rejection	E IONC NO TARGET ADDRESS FOUND FOR ROUTING PARAMETERS
SOAP Fault with Delivery Rejection	E ION OPERATION NOT IMPLEMENTED
SOAP Fault with Delivery Rejection	E ION ORIGINAL REQUEST NOT FOUND
SOAP Fault with Delivery Rejection	E IONC RECEIVER NOT REACHABLE
SOAP Fault with Delivery Rejection	E IONC RECEIVER RESPONSE TIMEOUT
SOAP Fault with Delivery Rejection	E IONC_UNKNOWN_OPERATION
SOAP Fault with Delivery Rejection	E IONC UNKNOWN OR UNAUTHORISED RECEIVER
SOAP Fault with Delivery Rejection	E IONC_UNKNOWN_OR_UNAUTHORISED_SENDER
SOAP Fault with Delivery Rejection	E ION UNKNOWN TECHNICAL PROBLEM
SOAP Fault with Delivery Rejection	E ION WRONG SENDER ROLE
SOAP Fault with Delivery Rejection	E ION WRONG SENDER SERVICE
SOAP Fault with Delivery Rejection	E CRE SERVICE NOT CONFIGURED

4.6.3 ION message with WSS

Conceptual WSDL message as used in the ION and which, due to the recommendations of the WS-I, has the same name as the underlying top-level element for the XML-based message and the associated operation.

In order to clarify compliance with the recommendations of the WS-I, the name of all derived WSDL messages is written in lower case here - as with the concrete WSDL messages in the use cases.

The WSDL message has - also due to the recommendations of the WS-I - only one message part, the top-level element mentioned above.

Example:

- Operation: notifyEntitlementIssued
- WSDL message: notifyEntitlementIssued
- XML Top-Level Element: notifyEntitlementIssued
- Response: notifyEntitlementIssuedResponse
- Exception: notifyEntitlementIssuedException



For the response context, the operation name is completed by the suffix "Response" or "Exception".

The corresponding SOAP header is then bound in the WSDL via the binding, also the WS-policy for the WSDL message or the whole operation (request and response).

To simplify matters, we use the WSDL message as an element in the model, which is sent completely as a SOAP message via the ION and considers the WSDL binding including header and WSS parts.

The [Interfaces](#) package contains the WSDL based interfaces imported into the model.

The example shows a message with an [ionRoutingHeader](#) and applied [Appendix: etiCORE Standard-Policy](#).

In order to have a coherent message example, the effects of MTOM are not shown here.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header xmlns:ion="https://eticore.org/ion/communication/3">
    <!-- Not for synchronous responses and business exceptions-->
    <ionRoutingHeader xmlns="https://eticore.org/ion/communication/3">
      <senderId>6598</senderId>
      <senderRole>1</senderRole>
      <senderService>ccp</senderService>
      <messageNumber>4711</messageNumber>
      <!--Optional, set only if greater 0:-->
      <repeatCounter>1</repeatCounter>
      <processInstanceId>SetServiceAvailable_5f7d9b47-b434-416a-9c60-adfde71dec1e</processInstanceId>
      <receiverId>5602</receiverId>
      <receiverRole>254</receiverRole>
      <receiverService>cre</receiverService>
      <interfaceVersion>3</interfaceVersion>
      <operationName>setServiceAvailable</operationName>
      <!--Optional:-->
      <optionalRoutingInfo>for regional purposes</optionalRoutingInfo>
    </ionRoutingHeader>
  <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
    soap:mustUnderstand="1">
    <wsu:Timestamp wsu:Id="TS-1e340445-f7f9-408a-8c0d-f8f6a4be4be9">
      <wsu:Created>2022-08-22T09:58:16.423Z</wsu:Created>
      <wsu:Expires>2022-08-22T10:03:16.423Z</wsu:Expires>
    </wsu:Timestamp>
    <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" Id="EK-f1763bae-1c1a-45b0-8f5a-70cc927818cb">
      <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p"/>
      <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <wsse:SecurityTokenReference>
          <ds:X509Data>
            <ds:X509IssuerSerial>
              <ds:X509IssuerName>CN=VDV Level 2 ION CA 2,O=VDV eTicket Service GmbH & Co. KG,C=DE</ds:X509IssuerName>
              <ds:X509SerialNumber>332756131495662</ds:X509SerialNumber>
            </ds:X509IssuerSerial>
          </ds:X509Data>
        </wsse:SecurityTokenReference>
      </ds:KeyInfo>
      <xenc:CipherData>
        <xenc:CipherValue>
          <!-- contains the encrypted symmetric key used for encryption of signature and body -->
        </xenc:CipherValue>
      </xenc:CipherData>
      <xenc:ReferenceList>
        <xenc:DataReference URI="#ED-2c8f3615-6498-4104-860f-035f1ae21883"/>
        <xenc:DataReference URI="#ED-9b6616fd-0851-46d4-9bbb-fab3b45195ce"/>
      </xenc:ReferenceList>
    </xenc:EncryptedKey>
    <xenc:EncryptedData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" Id="ED-9b6616fd-0851-46d4-9bbb-fab3b45195ce"
      Type="http://www.w3.org/2001/04/xmlenc#Element">
      <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes256-cbc"/>
      <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
```

```
<wsse:SecurityTokenReference
  xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
  xmlns:wsse11="http://docs.oasis-open.org/wss/oasis-wss-wssecurity-secext-1.1.xsd"
  wsse11:TokenType="http://docs.oasis-open.org/wss/oasis-wss-soap-message-security-1.1#EncryptedKey">
  <wsse:Reference URI="#EK-f1763bae-1c1a-45b0-8f5a-70cc927818cb"/>
</wsse:SecurityTokenReference>
</ds:KeyInfo>
<xenc:CipherData>
  <xenc:CipherValue>
    <!--
      Contains the encrypted signature and its metadata. The decrypted content would look like this:
      see Signature
    -->
  </xenc:CipherValue>
</xenc:CipherData>
</xenc:EncryptedData>
</wsse:Security>
</soap:Header>
<soap:Body xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
  wsu:Id="_f2420367-bdff-4538-a8ab-98597c395e2e">
  <xenc:EncryptedData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" Id="ED-2c8f3615-6498-4104-860f-
035f1ae21883"
    Type="http://www.w3.org/2001/04/xmlenc#Content">
    <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes256-cbc"/>
    <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      <wsse:SecurityTokenReference
        xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
        xmlns:wsse11="http://docs.oasis-open.org/wss/oasis-wss-wssecurity-secext-1.1.xsd"
        wsse11:TokenType="http://docs.oasis-open.org/wss/oasis-wss-soap-message-security-1.1#EncryptedKey">
        <wsse:Reference URI="#EK-f1763bae-1c1a-45b0-8f5a-70cc927818cb"/>
      </wsse:SecurityTokenReference>
    </ds:KeyInfo>
    <xenc:CipherData>
      <xenc:CipherValue>
        <!-- Contains the encrypted body -->
      </xenc:CipherValue>
    </xenc:CipherData>
  </xenc:EncryptedData>
</soap:Body>
</soap:Envelope>
```

4.6.4 ION reply message

Conceptual WSDL message as used in the ION for all replies: responses or (business) exceptions.

4.6.5 ION request message

This element defines a conceptual [ION request message](#) as a placeholder for all concrete request messages used in the ION.

Caution: an [ION request message](#) can be used in a synchronous and asynchronous context. If a synchronous response is expected for the request, the request has a synchronous context. The context itself is defined inside the CRE.

The ION request message itself does not differ in these different contexts. [SOAP header](#) and message composition are the same for both contexts.

The example is shown without WSS for better understanding. The real message would look like the [ION SOAP message with WSS](#).

For the composition of an ION request message, see [Supporting Objects : ION request message composition](#).

The diagram shows an incoming [ION message](#) represented by a [SOAP Envelope](#) and the related parts.

The diagram shows the composition of the non-encrypted [SOAP Header](#) with its etiCORE header information, as well as the [Security Header](#). The content is a result of the etiCORE WSS policy. The SOAP body contains the encrypted payload in form of a [request payload](#).

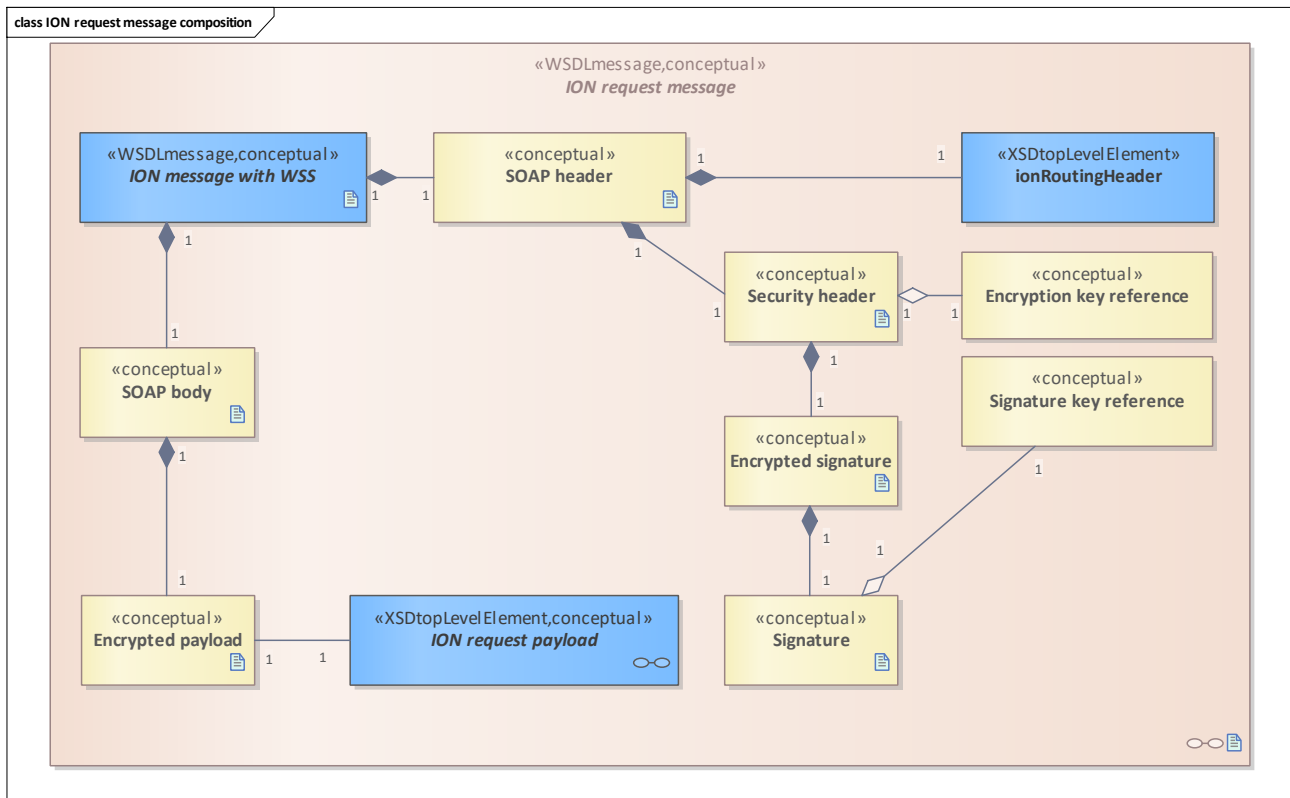


Figure 19: ION request message composition

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <ionRoutingHeader xmlns="https://eticore.org/ion/communication/3">
      <senderId>6598</senderId>
      <senderRole>1</senderRole>
      <senderService>ccp</senderService>
      <messageNumber>ccp-4710</messageNumber>
      <!--Optional, set only if greater 0:-->
      <repeatCounter>1</repeatCounter>
      <processInstanceId>[To xxx related basic process]_[UUID]</processInstanceId>
      <receiverId>42</receiverId>
      <receiverRole>3</receiverRole>
      <receiverService>po</receiverService>
      <interfaceVersion>3.0.0</interfaceVersion>
      <operationName>xxx</operationName>
      <!--Optional:-->
      <optionalRoutingInfo>for regional purposes</optionalRoutingInfo>
    </ionRoutingHeader>
    <wsse:Security xmlns:wsse="...">
      <!-- if WSS is active, the Security header is placed here. -->
    </wsse:Security>
  </soap:Header>
  <soap:Body>
    <!-- if WSS is active, the body is encrypted. -->
    <po:xxx xmlns="https://eticore.org/ion/3" xmlns:po="https://eticore.org/po/messaging/3">
      <!-- Top-level element xxx instantiates the related data type Xxx
      that extends Request -->
      <communicationInformation>
        <receiverId>42</receiverId>
        <receiverRole>3</receiverRole>
        <receiverService>po</receiverService>
        <messageId>
          <senderId>6598</senderId>
          <senderRole>1</senderRole>
        </messageId>
      </communicationInformation>
    </po:xxx>
  </soap:Body>
</soap:Envelope>
```

```
<senderService>ccp</senderService>
<messageNumber>ccp-4710</messageNumber>
</messageId>
<messageTimestamp>2022-08-10T08:53:31.569+02:00</messageTimestamp>
<!--Optional, set only if greater 0:-->
<repeatCounter>1</repeatCounter>
<processInstanceId>[To xxx related basic process]_[UUID]</processInstanceId>
</communicationInformation>
<!-- here could be placed additional request business data depending on
the use case and the XSD that defines Xxx -->
</po:xxx>
</soap:Body>
</soap:Envelope>
```

4.6.6 ION synchronous response message

This element defines a conceptual ION synchronous response message as a placeholder for all concrete synchronous response messages used in the ION.

The example is shown without WSS for better understanding. The real message would look like the [ION SOAP message with WSS](#) without the [ionRoutingHeader](#).

For the composition of a synchronous response message, see [Synchronous ION response message composition](#).

The diagram shows a synchronous ION response message derived from an [ION SOAP message with WSS](#) and the related parts.

The diagram shows the composition of the non-encrypted [SOAP header](#) with its [security header](#). The content is a result of the WS policy (see [Appendix: etiCORE Standard-Policy](#)).

The response is characterised by the reference to the request which caused this response.

Furthermore, an optional list of events is provided in each response.

The message header of the response message differs from the one of a request message, since due to the synchronicity, routing information is not needed.

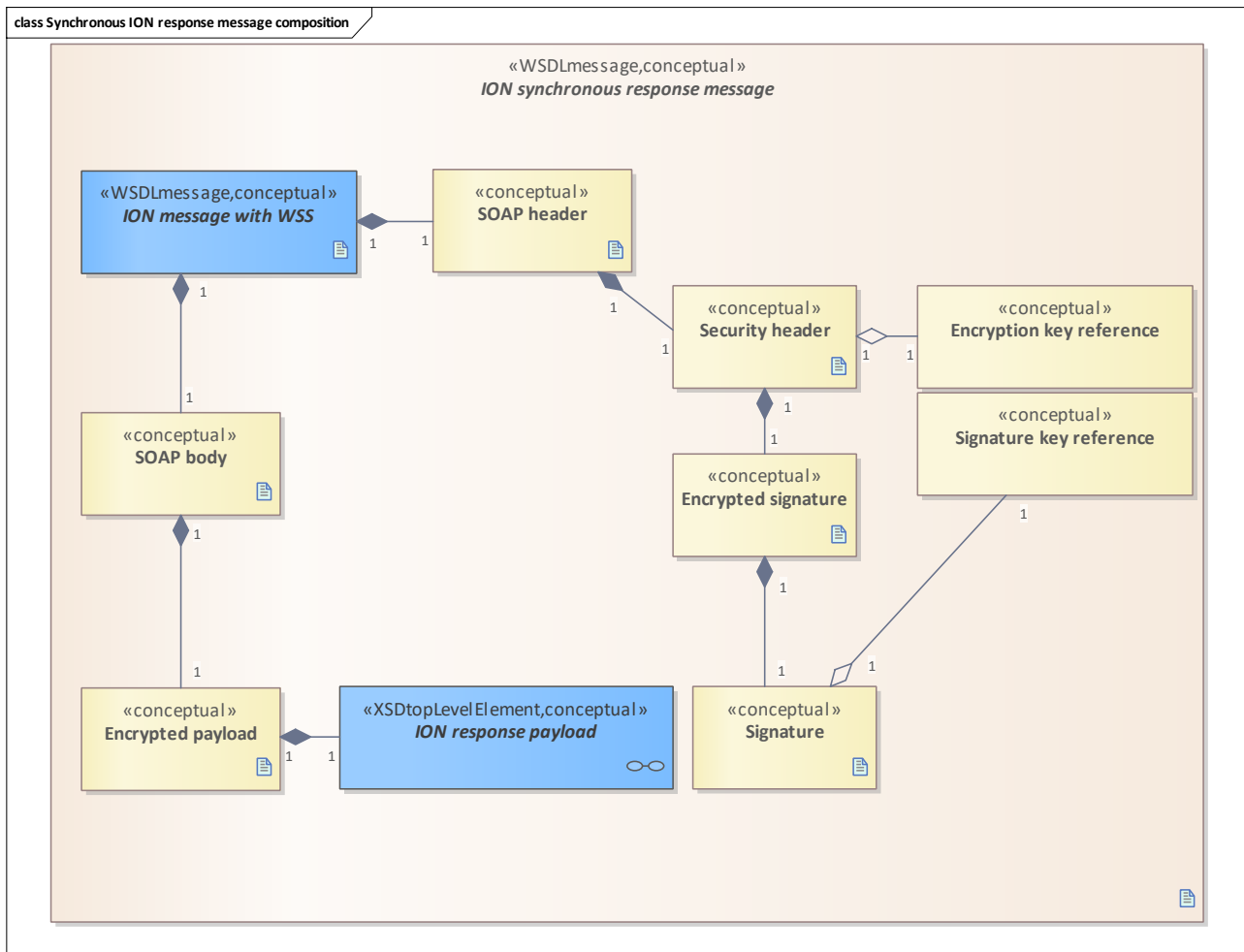


Figure 20: Synchronous ION response message composition

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <!-- no ionRoutingHeader in SOAP header since response is synchronous -->
  <soap:Header>
    <wsse:Security xmlns:wsse="...">
      <!-- if WSS is active, the Security header is placed here. -->
    </wsse:Security>
  </soap:Header>
  <soap:Body>
    <!-- if WSS is active, the body is encrypted. -->
    <po:xxxResponse xmlns="https://eticore.org/ion/3" xmlns:po="https://eticore.org/po/messaging/3">
      <!-- Top-level element xxxResponse instantiates the related data type XxxResponse
            that extends BusinessAcknowledgement -->
      <communicationInformation>
        <receiverId>6398</receiverId>
        <receiverRole>1</receiverRole>
        <receiverService>ccp</receiverService>
        <messageId>
          <senderId>42</senderId>
          <senderRole>3</senderRole>
          <senderService>po</senderService>
          <messageNumber>po-4711</messageNumber>
        </messageId>
        <messageTimestamp>2022-08-10T08:53:31.569+02:00</messageTimestamp>
        <!--Optional, set only if greater 0-->
        <repeatCounter>1</repeatCounter>
        <processInstanceId>[To xxx related basic process]_[UUID]</processInstanceId>
      </communicationInformation>
      <originalRequestCommunicationInformation>
        <originalRequestCorrelationId>
          <senderId>6398</senderId>
          <senderRole>1</senderRole>
        </originalRequestCorrelationId>
      </originalRequestCommunicationInformation>
    </po:xxxResponse>
  </soap:Body>
</soap:Envelope>
```

```
<senderService>ccp</senderService>
<messageNumber>ccp-4710</messageNumber>
</originalRequestCorrelationId>
<originalRequestTimestamp>2022-08-10T08:53:31.327+02:00</originalRequestTimestamp>
<!--Optional, only if set in request:-->
<originalRepeatCounter>1</originalRepeatCounter>
</originalRequestCommunicationInformation>
<!-- here could be placed additional response business data depending on
the use case and the XSD which defines XxxResponse -->
</po:xxxResponse>
</soap:Body>
</soap:Envelope>
```

4.6.7 ION synchronous exception message

This element defines a conceptual ION synchronous exception message as a placeholder for all concrete synchronous exception messages used in the ION.

The example is shown without WSS for better understanding. The real message would look like the [ION SOAP message with WSS](#) without the [ionRoutingHeader](#).

For the composition of a synchronous exception message, see [Supporting Objects : Synchronous ION exception message composition](#).

The diagram shows the composition of a synchronous ION exception message represented by a [SOAP envelope](#) and the related parts.

The diagram shows the composition of the non-encrypted [SOAP header](#) with its [Security header](#). The content is a result of the WS policy (see [Appendix: etiCORE Standard-Policy](#)).

The exception has a reference to the request which caused this exception and contains a unique string-based error code. Furthermore, an optional list of additional events is provided.

The message header of the response message differs from the one of a request message, since due to the synchronicity, routing information is not needed.

In the synchronous context, the [exception payload](#) is embedded into a [SOAP Fault](#).

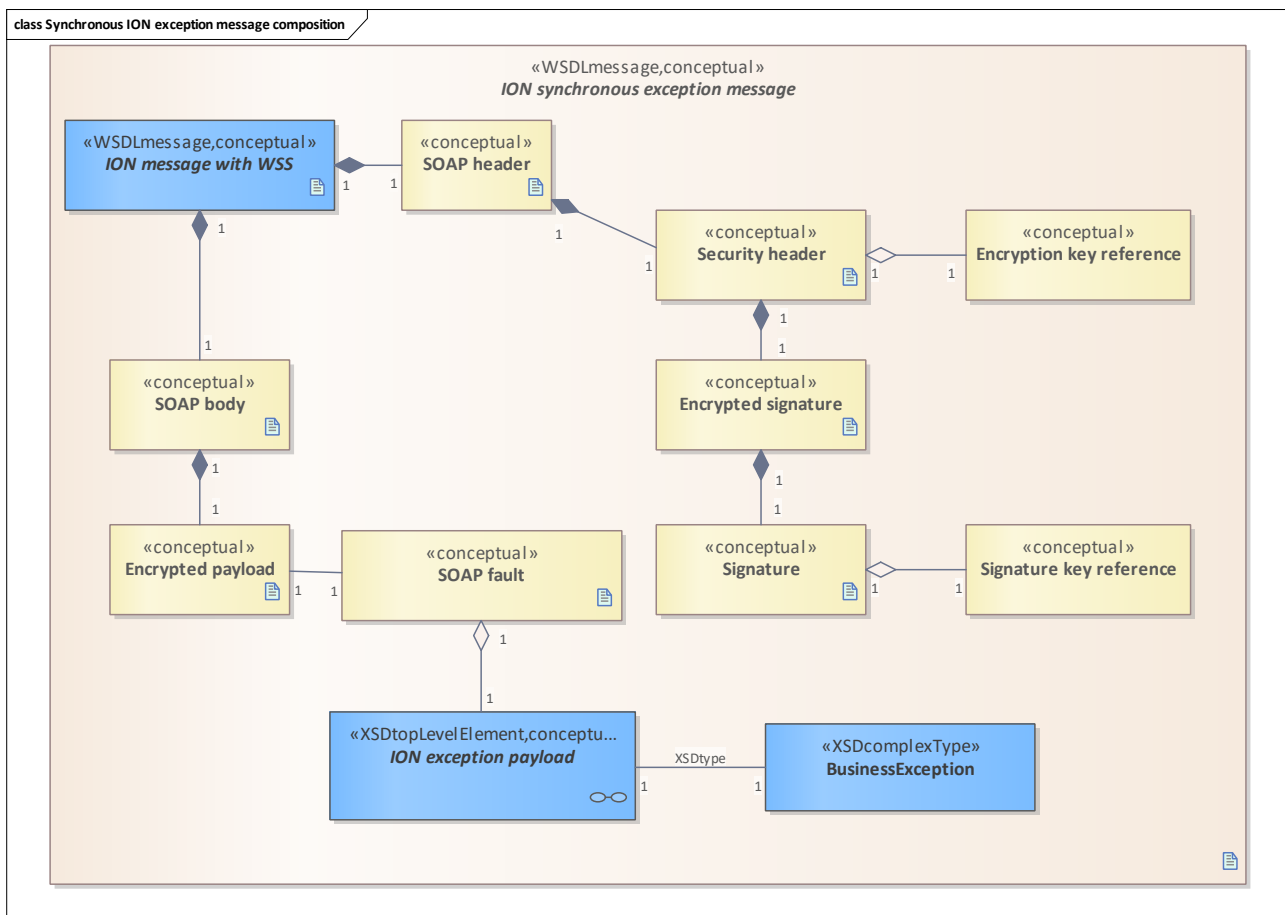


Figure 21: Synchronous ION exception message composition

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <!-- no ionRoutingHeader in SOAP header since response is synchronous -->
  <soap:Header>
    <wsse:Security xmlns:wsse="...">
      <!-- if WSS is active, the Security header is placed here. -->
    </wsse:Security>
  </soap:Header>
  <soap:Body>
    <!-- if WSS is active, the body is encrypted. -->
    <soap:Fault>

      <!-- Not specified in etiCORE. Field is mandatory
      Recommendation: use the predefined
      term "Server" if the system received the message,
      use the predefined term "Client" if the message
      did not leave the system (e.g. outgoing schema validation failed) -->
      <faultcode>soap:Server</faultcode>

      <!-- If a soap fault is thrown due to security reasons,
      leave the field empty (field is mandatory but nillable).
      For specified exceptions, the string of the detail section
      can be filled here -->
      <faultstring>E_HL_HOTLIST_ENTRY_ALREADY_EXISTS</faultstring>

      <detail>
        <!-- Top-level element addApplicationToHotlistException instantiates
        the data type BusinessException -->
        <hl:addApplicationToHotlistException xmlns:hl="https://eticore.org/hotlist/messaging/3"
        xmlns="https://eticore.org/ion/3">
          <communicationInformation>
            <receiverId>6398</receiverId>
            <receiverRole>1</receiverRole>
            <receiverService>ccp</receiverService>
            <messageId>
```

```
<senderId>5600</senderId>
<senderRole>5</senderRole>
<senderService>hotlist</senderService>
<messageNumber>hotlist-4711</messageNumber>
</messageId>
<messageTimestamp>2022-08-10T08:53:31.569+02:00</messageTimestamp>
<!--Optional, set only if greater 0:-->
<repeatCounter>1</repeatCounter>
<processInstanceId>AddApplicationToHostlist_[UUID]</processInstanceId>
</communicationInformation>
<originalRequestCommunicationInformation>
  <originalRequestCorrelationId>
    <senderId>6398</senderId>
    <senderRole>1</senderRole>
    <senderService>ccp</senderService>
    <messageNumber>ccp-4710</messageNumber>
  </originalRequestCorrelationId>
  <originalRequestTimestamp>2022-08-10T08:53:31.327+02:00</originalRequestTimestamp>
  <!--Optional, only if set in request:-->
  <originalRepeatCounter>1</originalRepeatCounter>
</originalRequestCommunicationInformation>
<error>
  <eventIdentifier>E_HL_HOTLIST_ENTRY_ALREADY_EXISTS</eventIdentifier>
  <eventDescription>The hotlist entry already exists in the hotlist service system</eventDescription>
</error>
</hl:addApplicationToHotlistException>
</detail>
</soap:Fault>
</soap:Body>
</soap:Envelope>
```

4.6.8 ION asynchronous response message

This element defines a conceptual ION asynchronous response message as a placeholder for all concrete asynchronous response messages used in the ION.

The example is shown without WSS for better understanding. The real message would look like the [ION SOAP message with WSS](#).

For the composition of an asynchronous response message, see [Supporting Objects : Asynchronous ION response message composition](#).

The diagram shows the composition of an [asynchronous ION response message](#) represented by a [SOAP Envelope](#) and the related parts.

The diagram shows the composition of the non-encrypted [SOAP Header](#) with its etiCORE header information, as well as the [Security header](#). The content is a result of the etiCORE WS policy (see [Appendix: etiCORE Standard-Policy](#)).

The response is characterised by the reference to the request which caused this response.

Furthermore, an optional list of events is provided in each response.

The message header of the response message does not differ in form from that of a request message, but the sender and recipient are swapped and the name of the response operation is inserted.

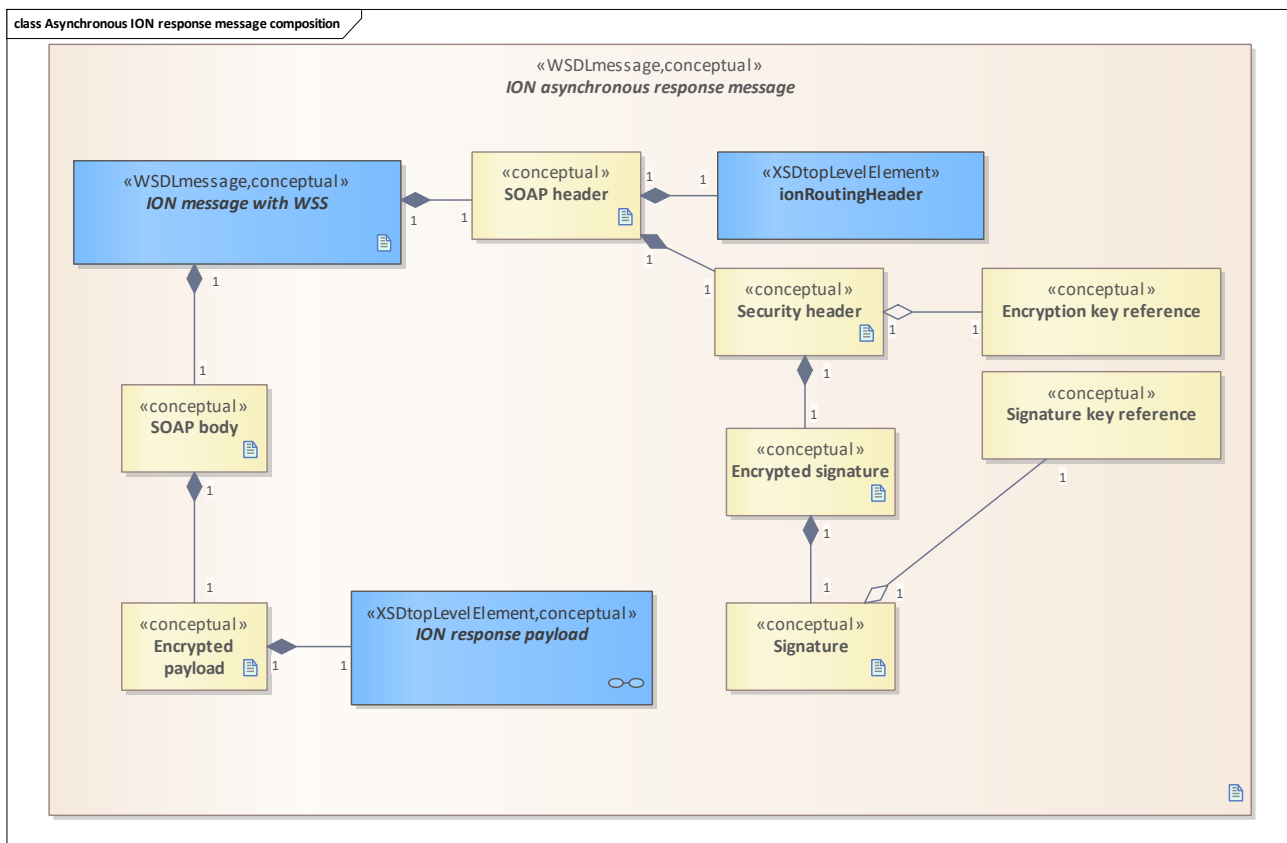


Figure 22: Asynchronous ION response message composition

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <ionRoutingHeader xmlns="https://eticore.org/ion/communication/3">
      <senderId>42</senderId>
      <senderRole>3</senderRole>
      <senderService>po</senderService>
      <messageNumber>po-4711</messageNumber>
      <!--Optional, set only if greater 0:-->
      <repeatCounter>1</repeatCounter>
      <processInstanceId>[To xxx related basic process]_[UUID]</processInstanceId>
      <receiverId>6598</receiverId>
      <receiverRole>1</receiverRole>
      <receiverService>ccp</receiverService>
      <interfaceVersion>3.0.0</interfaceVersion>
      <operationName>xxxResponse</operationName>
      <!--Optional:-->
      <optionalRoutingInfo>for regional purposes</optionalRoutingInfo>
    </ionRoutingHeader>
    <wsse:Security xmlns:wsse="...">
      <!-- if WSS is active, the Security header is placed here. -->
    </wsse:Security>
  </soap:Header>
  <soap:Body>
    <!-- if WSS is active, the body is encrypted. -->
    <ccp:xxxResponse xmlns="https://eticore.org/ion/3" xmlns:ccp="https://eticore.org/ccp/messaging/3">
      <!-- Top-level element xxxResponse instantiates the related data type XxxResponse
           that extends BusinessAcknowledgement -->
      <communicationInformation>
        <receiverId>6598</receiverId>
        <receiverRole>1</receiverRole>
        <receiverService>ccp</receiverService>
        <messageId>
          <senderId>42</senderId>
          <senderRole>3</senderRole>
          <senderService>po</senderService>
          <messageNumber>po-4711</messageNumber>
        </messageId>
      </communicationInformation>
    </ccp:xxxResponse>
  </soap:Body>
</soap:Envelope>
```

```
</messageId>
<messageTimestamp>2022-08-10T08:53:31.569+02:00</messageTimestamp>
<!--Optional, set only if greater 0:-->
<repeatCounter>1</repeatCounter>
<processInstanceId>[To xxx related basic process]_[UUID]</processInstanceId>
</communicationInformation>
<originalRequestCommunicationInformation>
  <originalRequestCorrelationId>
    <senderId>6598</senderId>
    <senderRole>1</senderRole>
    <senderService>ccp</senderService>
    <messageNumber>ccp-4710</messageNumber>
  </originalRequestCorrelationId>
  <originalRequestTimestamp>2022-08-10T08:53:31.327+02:00</originalRequestTimestamp>
  <!--Optional, only if set in request:-->
  <originalRepeatCounter>1</originalRepeatCounter>
</originalRequestCommunicationInformation>
<!-- here could be placed additional response business data depending on
the use case and the XSD which defines XxxResponse -->
</ccp:xxxResponse>
</soap:Body>
</soap:Envelope>
```

4.6.9 ION asynchronous exception message

This element defines a conceptual ION asynchronous exception message as a placeholder for all concrete asynchronous exception messages used in the ION.

The example is shown without WSS for better understanding. The real message would look like the [ION SOAP message with WSS](#).

For the composition of an asynchronous exception message, see [Supporting Objects : Asynchronous ION exception message composition](#).

The diagram shows the composition of an [asynchronous ION exception message](#) represented by a [SOAP envelope](#) and the related parts.

The diagram shows the composition of the non-encrypted [SOAP header](#) with its etiCORE header information, as well as the [security header](#). The content is a result of the WS policy (see [Appendix: etiCORE Standard-Policy](#)).

The exception has a reference to the request which caused this exception and contains a unique string-based error code. Furthermore, an optional list of additional events is provided.

The message header of the exception message does not differ in form from the one of a regular request or response.

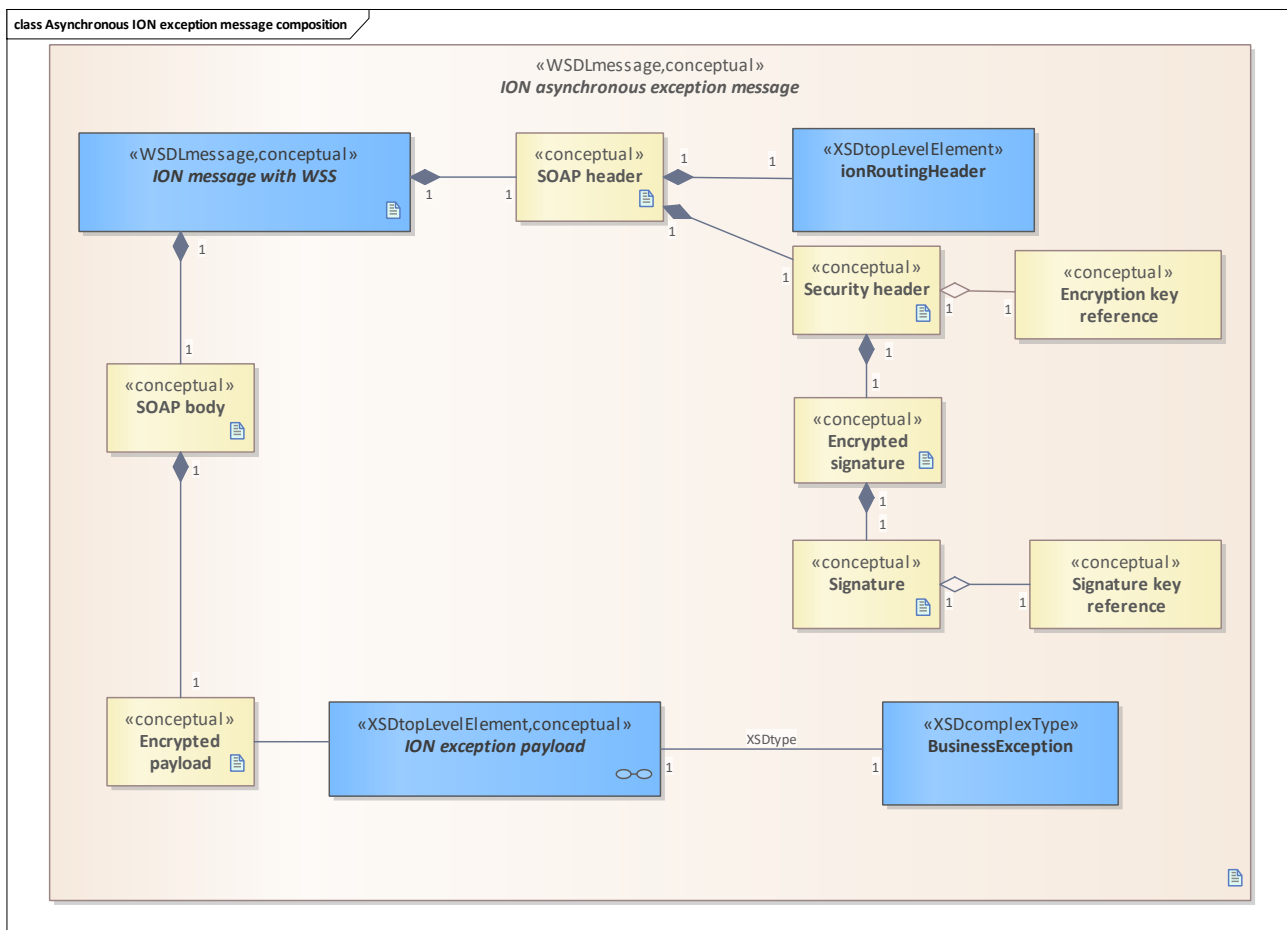


Figure 23: Asynchronous ION exception message composition

```

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <ionRoutingHeader xmlns="https://eticore.org/ion/communication/3">
      <senderId>42</senderId>
      <senderRole>3</senderRole>
      <senderService>po</senderService>
      <messageNumber>po-4711</messageNumber>
      <!--Optional, set only if greater 0:-->
      <repeatCounter>1</repeatCounter>
      <processInstanceId>[To xxx related basic process]_[UUID]</processInstanceId>
      <receiverId>6598</receiverId>
      <receiverRole>1</receiverRole>
      <receiverService>ccp</receiverService>
      <interfaceVersion>3.0.0</interfaceVersion>
      <operationName>xxxException</operationName>
      <!--Optional:-->
      <optionalRoutingInfo>for regional purposes</optionalRoutingInfo>
    </ionRoutingHeader>
    <wsse:Security xmlns:wsse="...">
      <!-- if WSS is active, the Security header is placed here. -->
    </wsse:Security>
  </soap:Header>
  <soap:Body>
    <!-- if WSS is active, the body is encrypted. -->
    <ccp:xxxException xmlns="https://eticore.org/ion/3" xmlns:ccp="https://eticore.org/ccp/messaging/3">
      <!-- Top-level element xxxException instantiates BusinessException -->
      <communicationInformation>
        <receiverId>6598</receiverId>
        <receiverRole>1</receiverRole>
        <receiverService>ccp</receiverService>
        <messageId>
          <senderId>42</senderId>
          <senderRole>3</senderRole>
        </messageId>
      </communicationInformation>
    </ccp:xxxException>
  </soap:Body>
</soap:Envelope>

```

```
<senderService>po</senderService>
<messageNumber>po-4711</messageNumber>
</messageId>
<messageTimestamp>2022-08-10T08:53:31.569+02:00</messageTimestamp>
<!--Optional, set only if greater 0:-->
<repeatCounter>1</repeatCounter>
<processInstanceId>[To xxx related basic process]_[UUID]</processInstanceId>
</communicationInformation>
<originalRequestCommunicationInformation>
  <originalRequestCorrelationId>
    <senderId>6598</senderId>
    <senderRole>1</senderRole>
    <senderService>ccp</senderService>
    <messageNumber>ccp-4710</messageNumber>
  </originalRequestCorrelationId>
  <originalRequestTimestamp>2022-08-10T08:53:31.327+02:00</originalRequestTimestamp>
  <!--Optional, only if set in request:-->
  <originalRepeatCounter>1</originalRepeatCounter>
</originalRequestCommunicationInformation>
<error>
  <eventIdentifier>E_CO_BINARY_STRUCTURE_CANNOT_BE_PROCESSED</eventIdentifier>
  <eventDescription>The binary structure contained in the request cannot be processed</eventDescription>
</error>
</ccp:xxxException>
</soap:Body>
</soap:Envelope>
```

4.6.10 ION message payload

(Abstract) payload of an ION message. For some situations, the payload type does not matter (e.g. the common steps when receiving a message).

The diagram shows the general composition of an ION payload. A payload can be either a part of a business request or a business response. A business response itself can be a regular response in the form of an (extended) [BusinessAcknowledgement](#) or a [BusinessException](#).

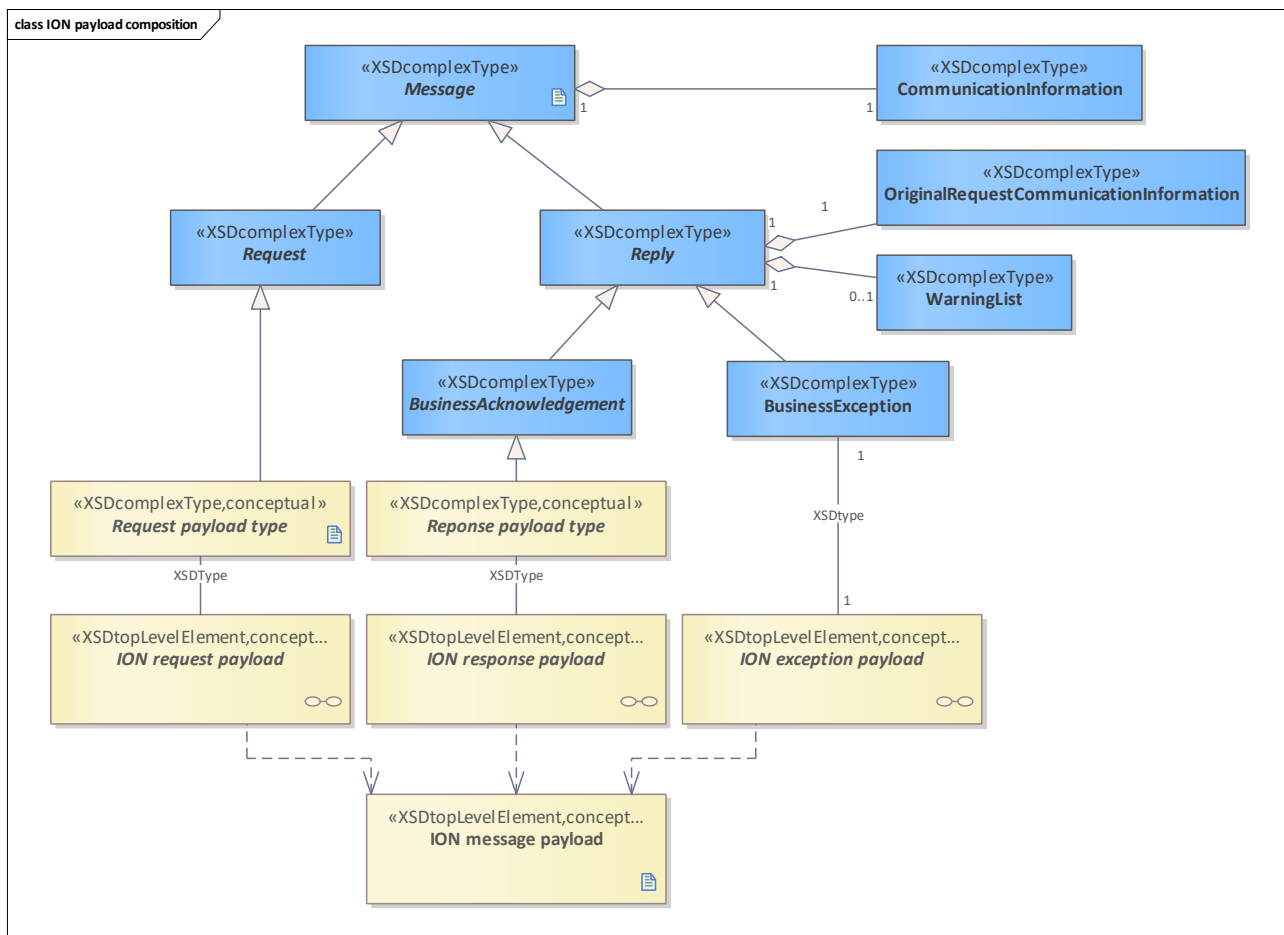


Figure 24: ION payload composition

4.6.11 ION request payload

This class represents the conceptual payload part of a request message and instantiates [Request payload type](#).

As a top-level element, it is a placeholder for all business requests sent via the ION. It is embedded in the SOAP body and due to the etiCORE WSS rules, completely encrypted. In order to clarify compliance with the recommendations of the WS-I, the name is written in lower case here - as with the concrete requests - which is the same as the name of the associated operation that processes this request.

The request payload consists of the [CommunicationInformation](#) and optionally contains an additional [Request business data](#) part.

The diagram shows the composition of an abstract request payload including an example.

A concrete request payload is always defined as a top-level element which points to a top-level type with the same name. This type inherits from [Request](#) which, in turn, inherits from [Message](#) that contains a [CommunicationInformation](#) part. The request payload is always named after the original operation (e.g. notifyXX).

Furthermore, an optional part of business data may be contained in form of [Request business data](#).

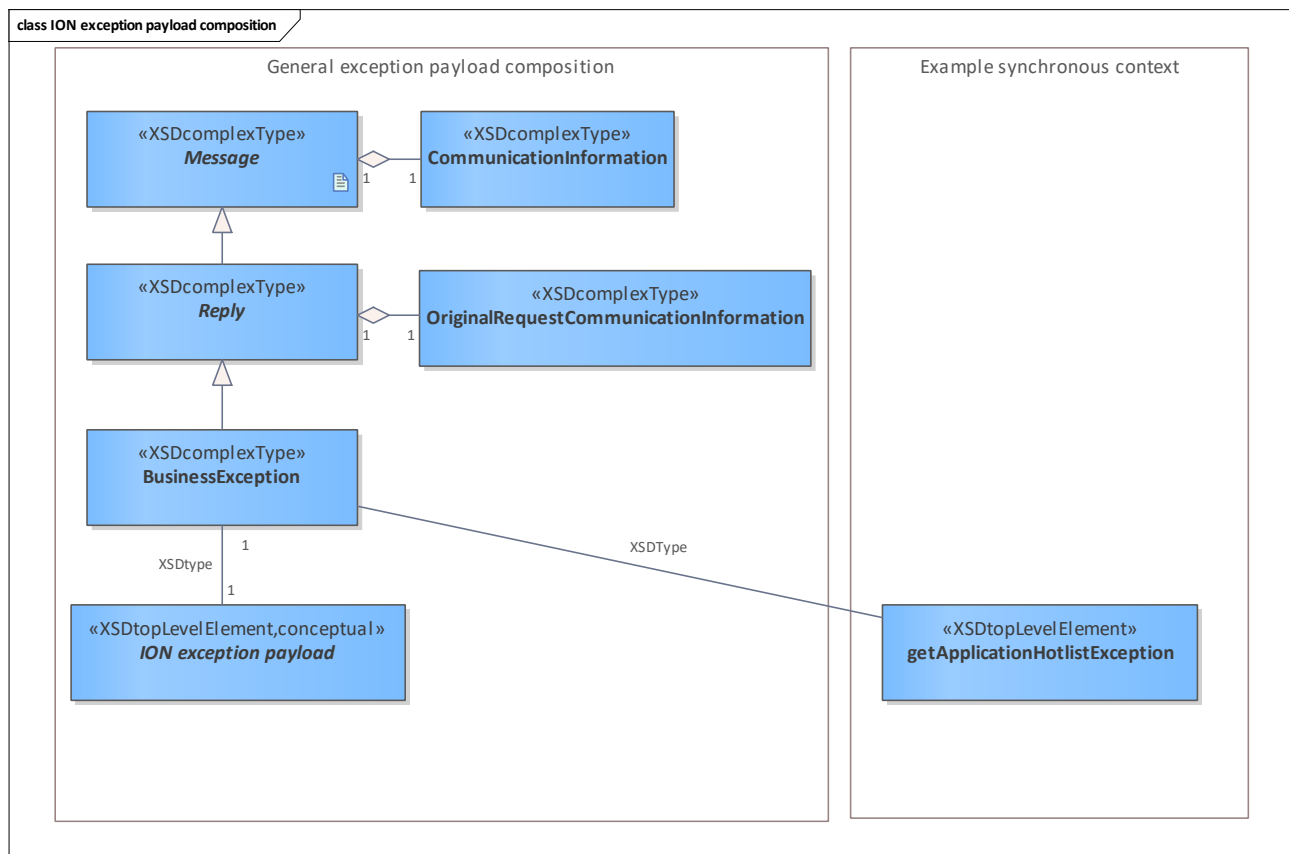


Figure 26: ION exception payload composition

4.6.13 Temporarily storable ION message

Conceptual WSDL message element that indicates that a corresponding ION message can be temporarily stored inside the CRE. Only certain types of messages are allowed to be stored inside the CRE:

- [ION request message](#) (in asynchronous context)
- [asynchronous ION response message](#)
- [asynchronous ION exception message](#)

The diagram shows the hierarchy of the possible message types which are suitable for store & forward in the ION environment. Store & forward is only possible in the asynchronous context.

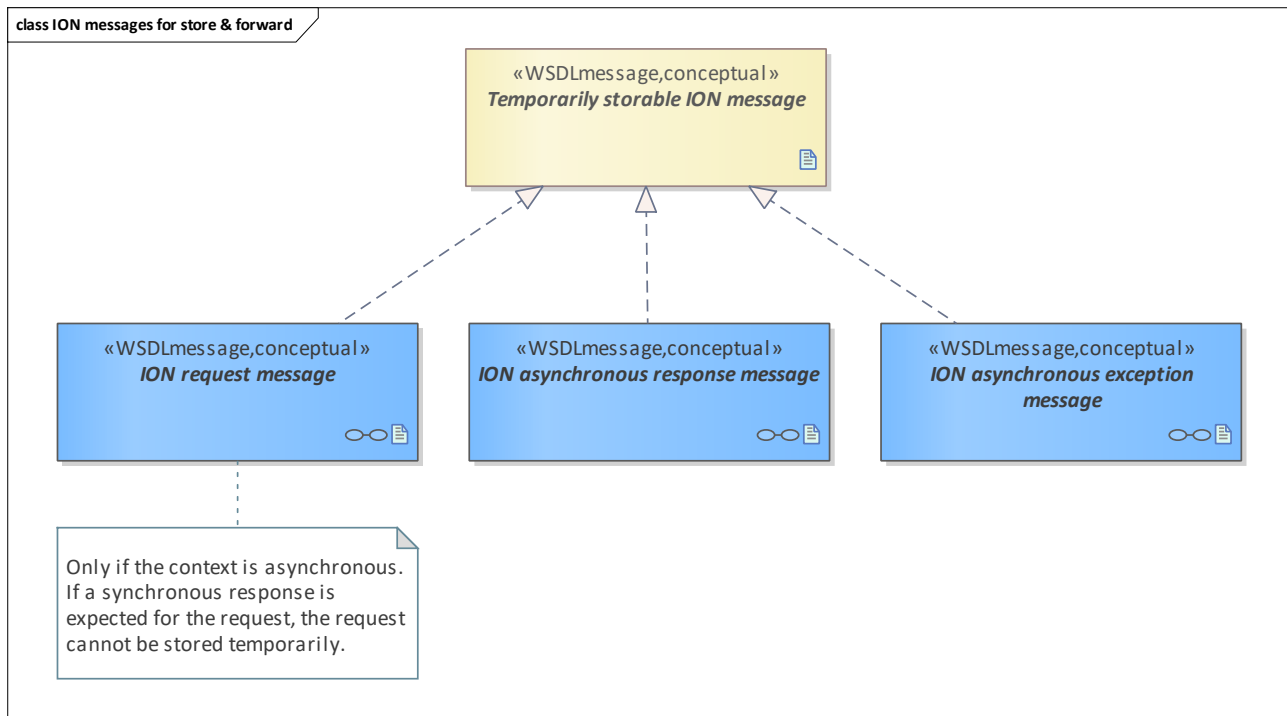


Figure 27: ION messages for store & forward

4.6.14 Delivery Acknowledgement

Direct reply for the asynchronous use case context.

Either sent from the [Processor](#) to the [Initiator](#) after incoming requests or from the [Initiator](#) to the [Processor](#) after incoming replies.

In case that the CRE stores the message, the CRE sends a [deliveryAcknowledgement](#) with `<deliveredTo>CRE</deliveredTo>`

Example Delivery Acknowledgement

Only sent if the receiver accepts the message or the CRE stored the message successfully. Note: the [deliveryAcknowledgement](#) does not work with web service security. The policy omits this message. Thus, the message is not encrypted and not signed.

```

<soap:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns="https://eticore.org/ion/communication/3">
  <soap:Header/>
  <soap:Body>
    <ns:deliveryAcknowledgement>
      <deliveredTo>CRE</deliveredTo>
    </ns:deliveryAcknowledgement>
  </soap:Body>
</soap:Envelope>

```

4.6.15 Delivery Rejection

Direct (error) reply for the asynchronous use case context wrapped with a SOAP fault.

Either sent from the [Processor](#) to the [Initiator](#) after incoming requests or from the [Initiator](#) to the [Processor](#) after incoming replies.

In case that the CRE rejects the message, the CRE sends a SOAP fault containing a [deliveryRejection](#) with `<deliveredTo>CRE</deliveredTo>`.

Example Delivery Rejection

Always sent in case of any error concerning delivery by receiver and CRE. Note: the [SOAP Fault](#) with [deliveryRejection](#) does not work with webservice security. The policy omits this message. Thus, the message is not encrypted and not signed.

```
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
    http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns="https://eticore.org/ion/communication/3">
  <soap:Body>
    <soap:Fault>

      <!--Not specified in etiCORE. Field is mandatory
        Recommendation: use the predefined
        term "Server" if the system received the message,
        use the predefined term "Client" if the message
        did not leave the system (e.g. outgoing schema validation failed)-->
      <faultcode>soap:Server</faultcode>

      <!--If a soap fault is thrown due to security reasons,
        leave the field empty (field is mandatory but nillable).
        For specified exceptions, the string of the detail section
        can be filled here.
      -->
      <faultstring>E_IONC_UNKNOWN_OPERATION</faultstring>

      <detail>
        <ns:deliveryRejection>
          <deliveredTo>CRE</deliveredTo>
          <ns:errorIdentifier>
            E_IONC_UNKNOWN_OPERATION
          </ns:errorIdentifier>
          <ns:errorText><!--Optional text message. CRE will not fill it--></ns:errorText>
        </ns:deliveryRejection>
      </detail>
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

4.6.16 SOAP fault

An element that transports a generic fault from the server to the client.

- In the synchronous context of messaging use case, a generic SOAP fault can be sent from the [Processor](#) to the [Initiator](#) for errors if a business exception is not (yet) possible
- In the synchronous context of the CRE use cases, the SOAP fault coming from the CRE transports communication error information using this generic SOAP fault.
- For runtime errors the web service frameworks use SOAP faults (out of specification scope)

Note:

- In the synchronous context of messaging use cases, the SOAP fault wraps the payload with the business exception data (see [ION synchronous exception message](#)), sent from the [Processor](#) to the [Initiator](#). The generic SOAP fault is not used
- In the asynchronous context, the SOAP fault transports communication error information using a [Delivery Rejection](#). This is either sent from the [Processor](#) to the [Initiator](#) (after incoming request) or from the [Initiator](#) to the [Processor](#) (after incoming reply). The generic SOAP fault is not used.

```
<soap:Envelope>
  <soap:Body>
    <soap:Fault>
```

```
<!-- Not specified in etiCORE. Field is mandatory
Recommendation: use the predefined
term "Server" if the system received the message,
use the predefined term "Client" if the message
did not leave the system (e.g. outgoing schema validation failed) -->
<faultcode>soap:Server</faultcode>
```

```
<!-- If a soap fault is thrown due to security reasons,
leave the field empty (field is mandatory but nillable).
For specified exceptions, the string of the error code
can be filled here -->
<faultstring>E_IONC_INVALID_SOAP_HEADER</faultstring>
```

```
</soap:Fault>
</soap:Body>
</soap:Envelope>
```

5 System Components and Interfaces

This chapter shows the system components, their interfaces and the realisation and usage of these interfaces.

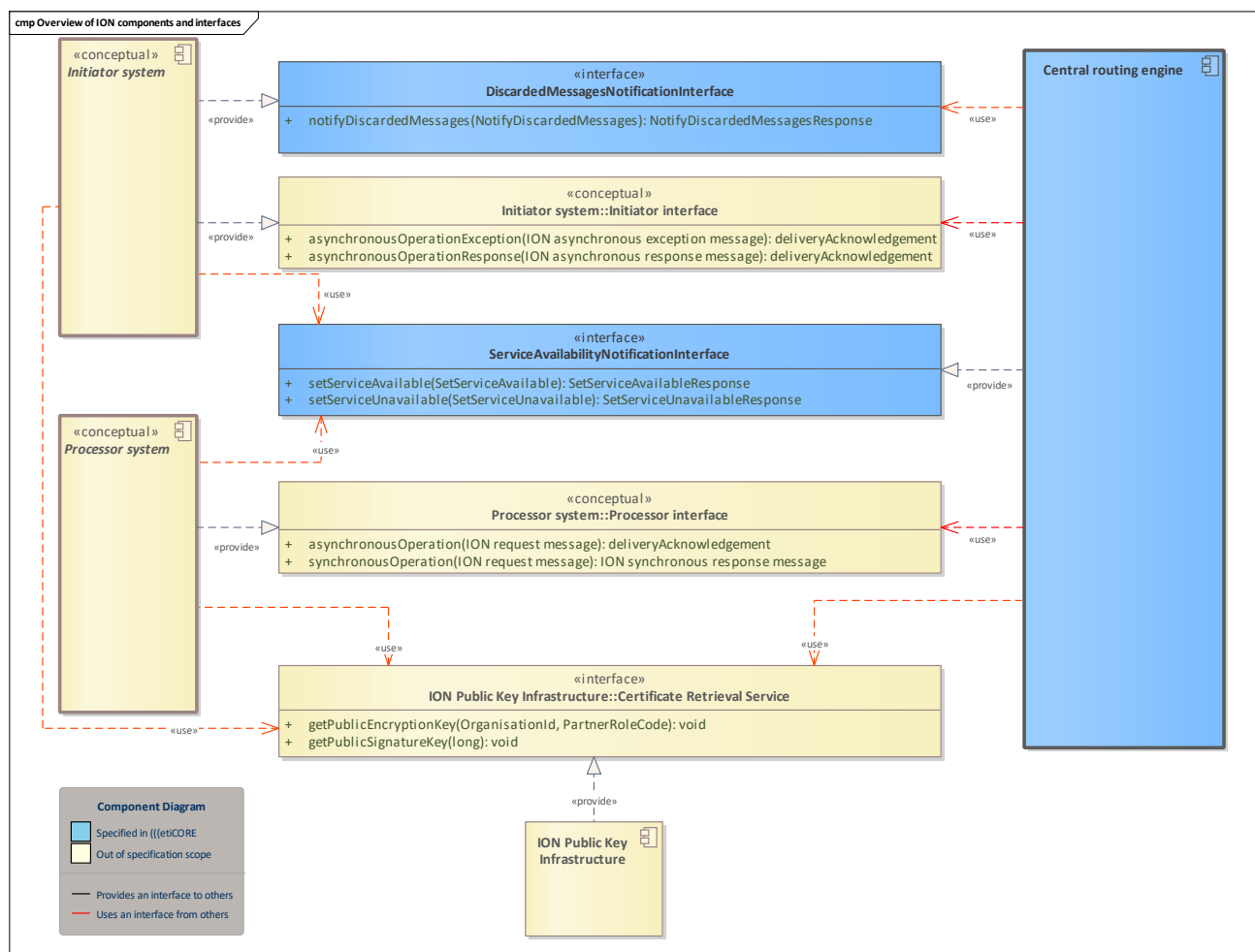


Figure 28: Overview of ION components and interfaces

This diagram shows the communication between the [Central routing engine](#), [Initiator system](#), [Processor system](#) and the [ION Public Key Infrastructure](#) via interfaces.

5.1 ION Public Key Infrastructure

PKI which stores the certificates and public keys for ION message exchange purposes:

- public encryption keys
- public signature keys
- TLS signature keys

5.1.1 Certificate Retrieval Service

Explanatory Interface that allows access to the public key of the [OrganisationalUnit](#) as [Initiator](#) for encryption and the signing key as [Processor](#) to verify the signature of messages and the transport layer.

Operations

Name: `getPublicEncryptionKey`

Description: Imaginary method to get a public key for message encryption by the [OrganisationId](#) and [PartnerRoleCode](#).

Name: `getPublicSignatureKey`

Description: Imaginary method to get a public signature key by its serial number.

5.2 Central routing engine

The component which implements the [Central routing engine](#).

Several interfaces are implemented and functionality provided:

- Functionality for store & forward
- Functionality for routing
- Interface for service availability notifications: [ServiceAvailabilityNotificationInterface](#)
- Conceptual interface as proxy for synchronous and asynchronous operations of [Processor](#) and [Initiator](#) systems

5.2.1 Central routing engine conceptual interface

Conceptual interface of the CRE only needed for explanatory purposes in sequence diagrams.

Operations

Name: `asynchronousOperation`

Description: Conceptual operation which provides the reception of an asynchronous request sent by the initiator.

Name: `asynchronousOperationException`

Description: Conceptual operation which provides the reception of an asynchronous business exception sent by the processor.

Operations

Name: asynchronousOperationResponse

Description: Conceptual operation which provides the reception of an asynchronous response sent by the processor.

Name: synchronousOperation

Description: Conceptual operation which provides the reception of a synchronous request sent by the initiator. The response is sent directly.

5.3 Initiator system

A component that implements an imaginary system of an [Initiator](#).

5.3.1 Initiator interface

The imaginary interface of an [Initiator system](#).

The provided operations can be considered as a placeholder for all real operations in synchronous and asynchronous contexts.

Operations

Name: asynchronousOperationException

Description: Conceptual operation which provides the reception of an asynchronous business exception sent by the processor.

Name: asynchronousOperationResponse

Description: Conceptual operation which provides the reception of an asynchronous response sent by the processor.

5.4 Processor system

A component that implements an imaginary system of a [Processor](#).

5.4.1 Processor interface

The imaginary interface of a [Processor system](#).

The provided operations can be considered as a placeholder for all real operations in synchronous and asynchronous contexts.

Operations

Name: asynchronousOperation

Description: Conceptual operation which provides the reception of an asynchronous request sent by the initiator.

Operations

Name: synchronousOperation

Description: Conceptual operation which provides the reception of a synchronous request sent by the initiator. The response is sent directly.

6 Use Cases

The chapter contains all use cases for ION messaging and the central routing engine.

6.1 ION Messaging Use Cases

This package contains the general use cases in the ION specification.

These general use cases offer conceptual processes for synchronous and asynchronous contexts. Furthermore, the required general checks before or after the business logic of a concrete use case are shown, so that these steps are explained only once and do not have to be repeated for each concrete use case.

6.1.1 Overview ION Messaging Use Cases

This chapter shows an overview of the most important ION use cases and their relationships in synchronous and asynchronous contexts.

6.1.1.1 Synchronous ION Use Case - Overview

This chapter shows an overview of the most important ION use cases and their relationship in a synchronous use case context. A synchronous use case context is divided into a [Initiator](#) and a [Processor](#) side with use cases on both sides.

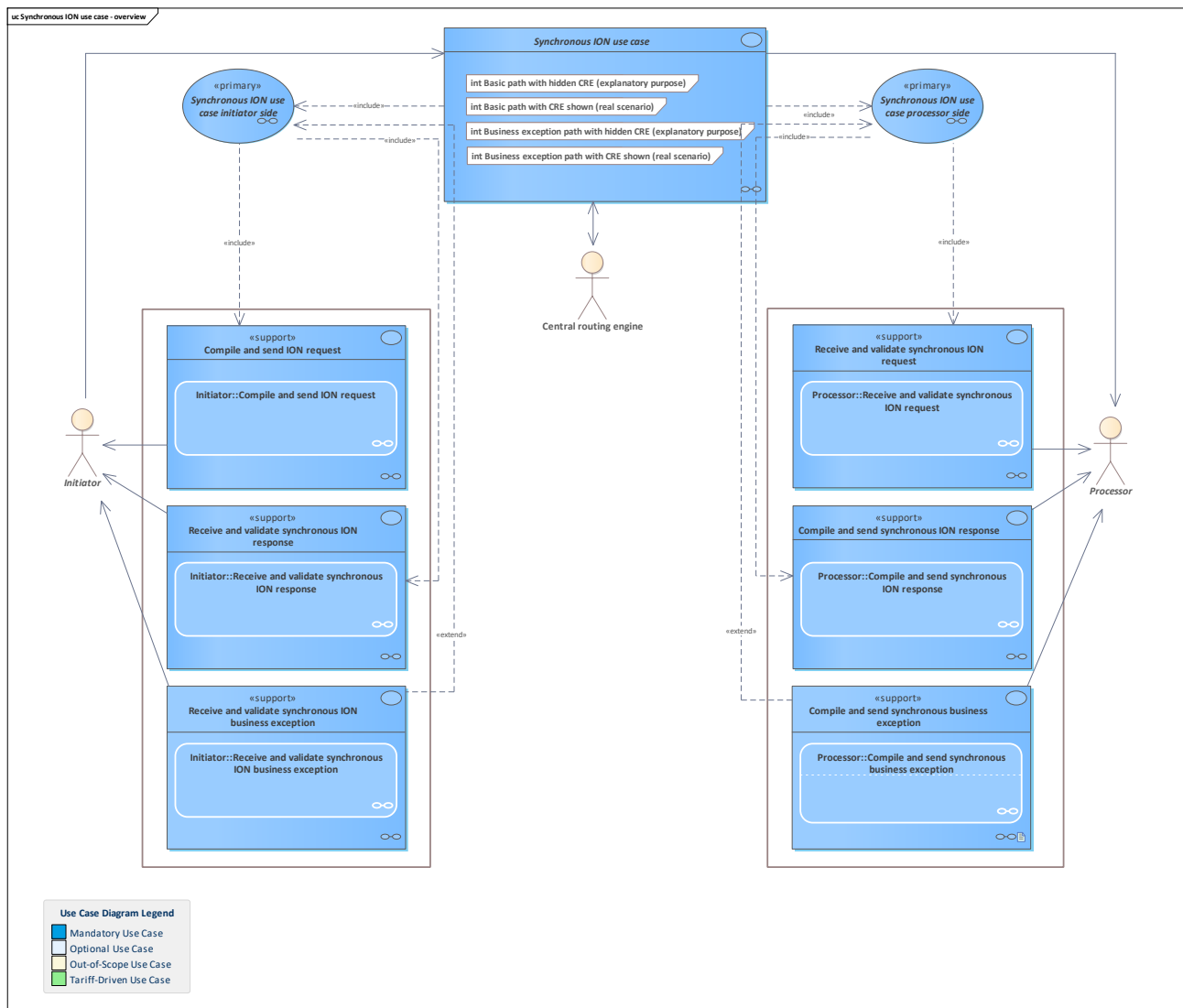


Figure 29: Synchronous ION use case - overview

See [Synchronous use case](#).

6.1.1.2 Asynchronous ION Use Case - Overview

This chapter shows an overview of the most important ION use cases and their relationship in an asynchronous use case context. An asynchronous use case context is divided into a [Initiator](#) and a [Processor](#) side with use cases on both sides.

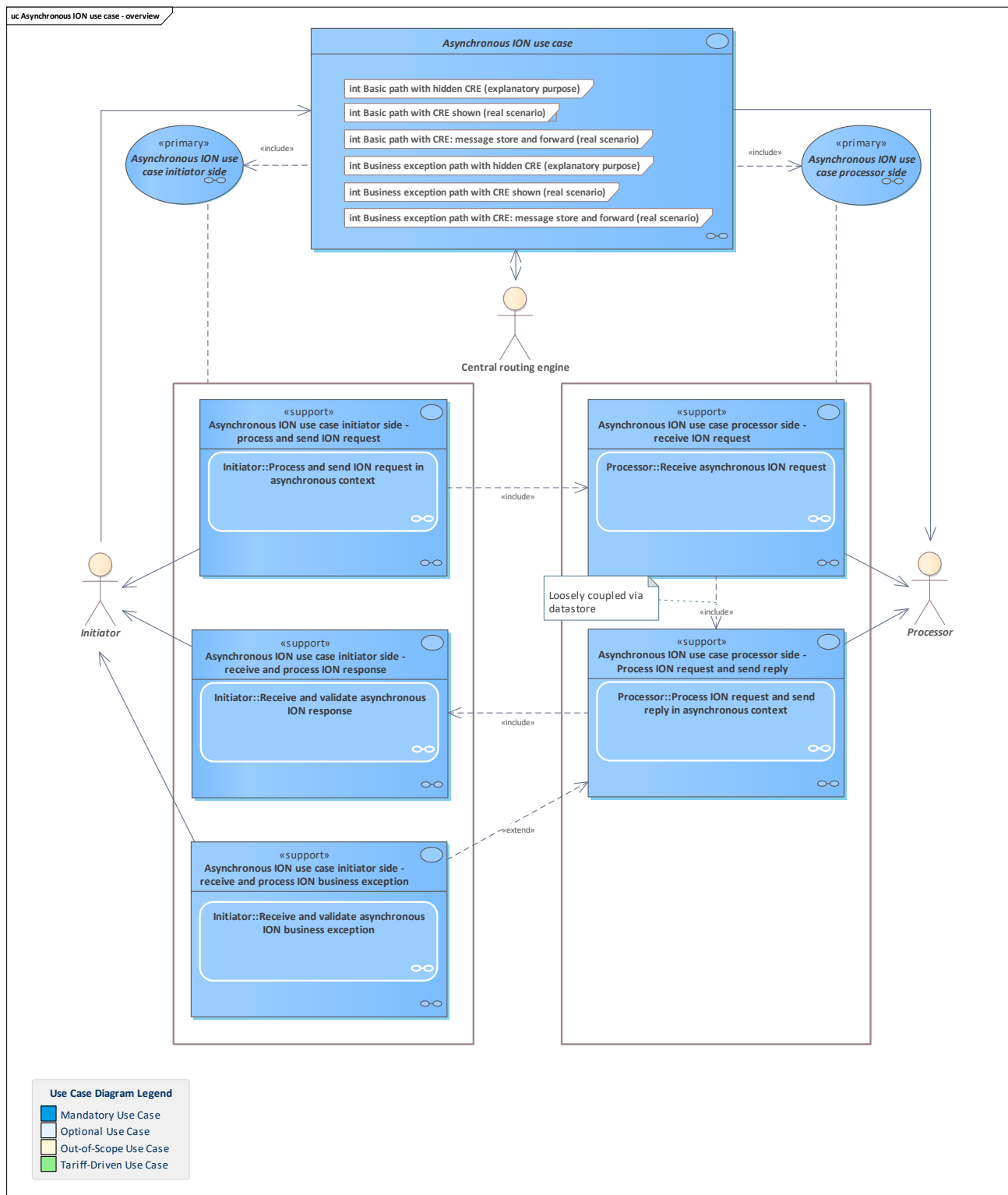
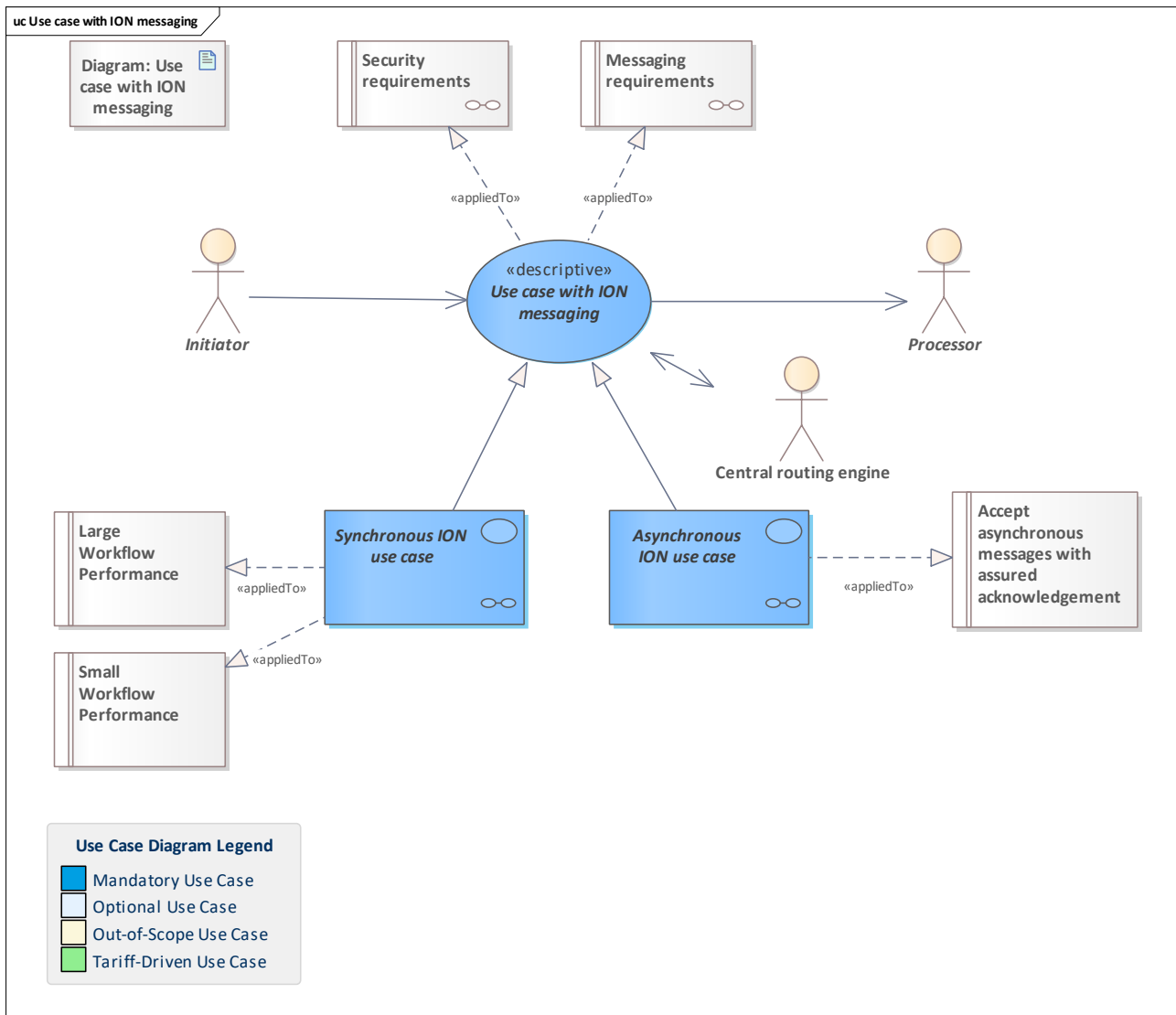


Figure 30: Asynchronous ION use case - overview

This diagram gives an overview for the supporting use cases that are included in an [Asynchronous ION use case](#).

6.1.1.3 Use case with ION messaging



Use Case	Use case with ION messaging
Description	<p>An abstract use case that involves message exchange in the ION and can be both synchronous or asynchronous. This use case has to fulfil or consider a lot of requirements:</p> <ul style="list-style-type: none"> ION:Security Messaging requirements Performance requirements <p>All derived use cases have to fulfil or consider these requirements.</p>
Initiating Actor	Initiator Central routing engine
Reacting Actor	Processor Central routing engine
Preconditions	
Postconditions	
Linked Use Cases (Extended By)	
Linked Use Cases (Includes)	
Linked Use Cases	Accept asynchronous messages with assured acknowledgement / www.eticket-deutschland.de

(Realises)	Large Workflow Performance / Small Workflow Performance / Security requirements / Messaging requirements
Base Activity	
Inputs	
Outputs	
Error Cases	
Activity Diagram	

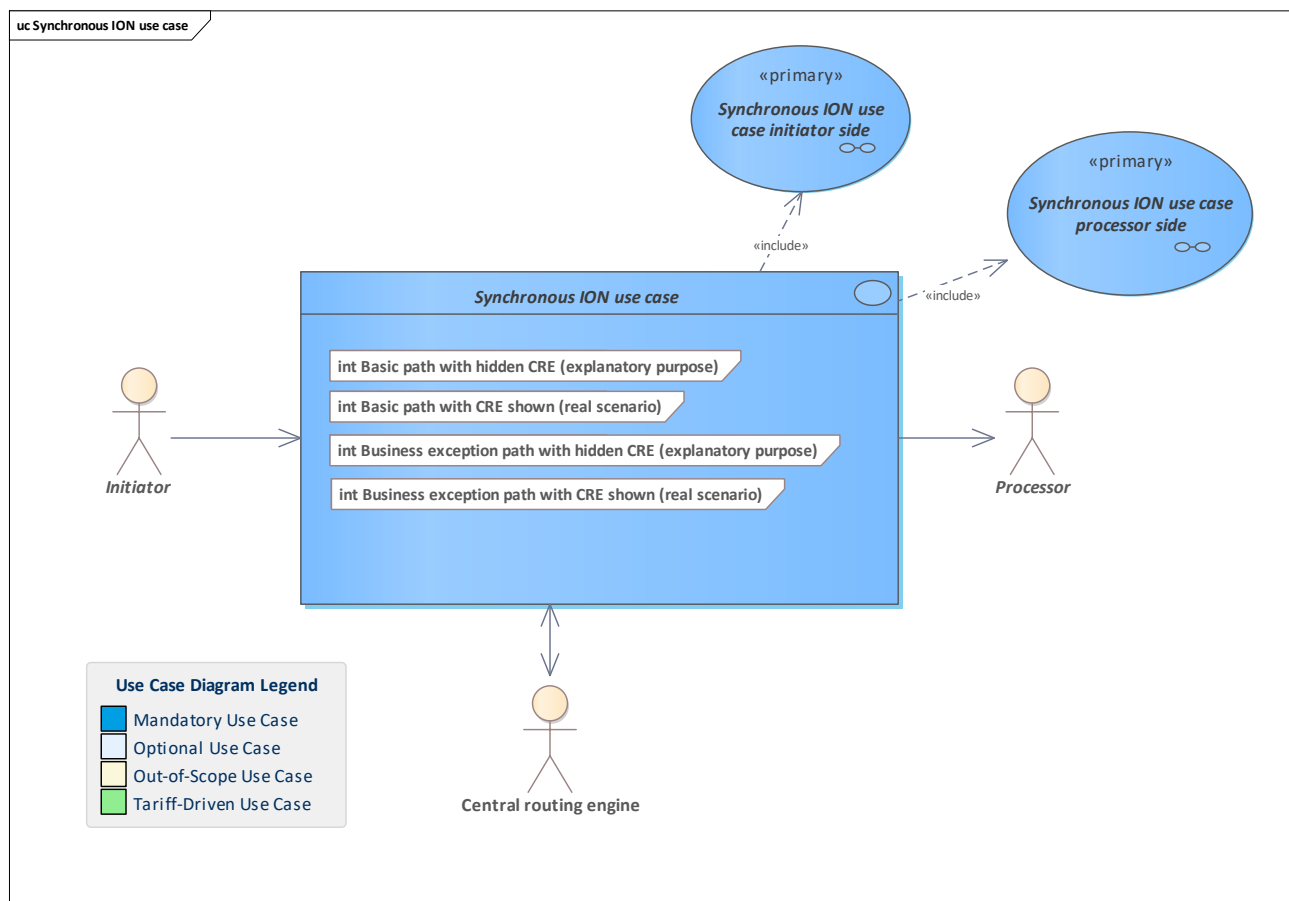
6.1.1.3.1 Diagram: Use case with ION messaging

The diagram shows two use cases that can occur in the context of ION messaging. One is the synchronous use case (see [Synchronous ION use case](#)) and the other is the asynchronous use case (see [Asynchronous ION use case](#)).

As a generic, abstract use case, [Use case with ION Messaging](#) stands for all use cases performing message exchange in the ION.

The use cases shown must each meet certain requirements. All derived, concrete use cases in the ION must therefore automatically also fulfil these requirements.

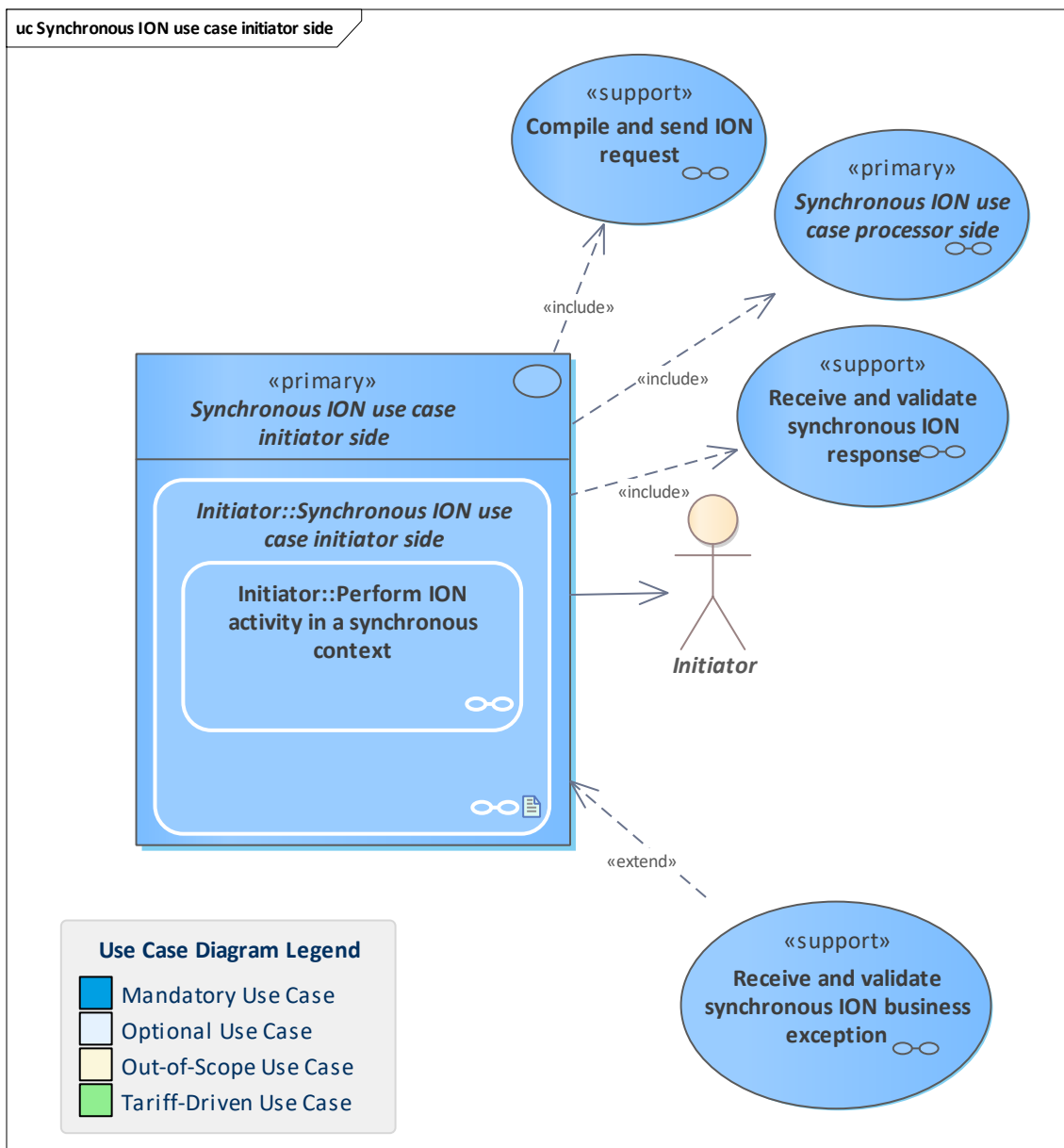
6.1.2 Synchronous ION use case



Use Case	Synchronous ION use case
Description	General use case which meets the requirements for a workflow

	<p>distributed over two systems involving only synchronous interactions.</p> <p>The synchronous use case can be divided into the side of the initiator and the side of the processor.</p> <p>Each synchronous use case works in the same way with a fixed workflow.</p> <p>The technical roles of client and server are never switched; the Initiator always remains the client while the Processor always remains the server. This behaviour is emphasized by the arrows in the associations between the actors and the use case.</p> <p>A synchronous use case means that message exchange driven by this use case is done in a synchronous context.</p> <ul style="list-style-type: none"> • The Initiator starts with a request and waits synchronously for a response • The Processor has all information to process this request directly and within the specified timeout interval and sends its response synchronously to the Initiator
Initiating Actor	Initiator Central routing engine
Reacting Actor	Processor Central routing engine
Preconditions	
Postconditions	
Linked Use Cases (Extended By)	
Linked Use Cases (Includes)	Synchronous ION use case initiator side / Synchronous ION use case processor side / Synchronous ION use case processor side
Linked Use Cases (Realises)	
Base Activity	
Inputs	
Outputs	
Error Cases	
Activity Diagram	

6.1.2.1 Synchronous ION use case initiator side



Use Case	Synchronous ION use case initiator side
Description	<p>This use case covers the whole initiator side in a synchronous use case started by the Initiator.</p> <p>The steps are:</p> <ul style="list-style-type: none"> Perform the business logic and collect business data for a request Build the suitable ION request and send it to the Processor Receive the synchronous ION response and do all generic checks Go through the remaining business logic to process the response depending on its content
Initiating Actor	
Reacting Actor	Initiator
Preconditions	
Postconditions	
Linked Use Cases (Extended By)	Receive and validate synchronous ION business exception

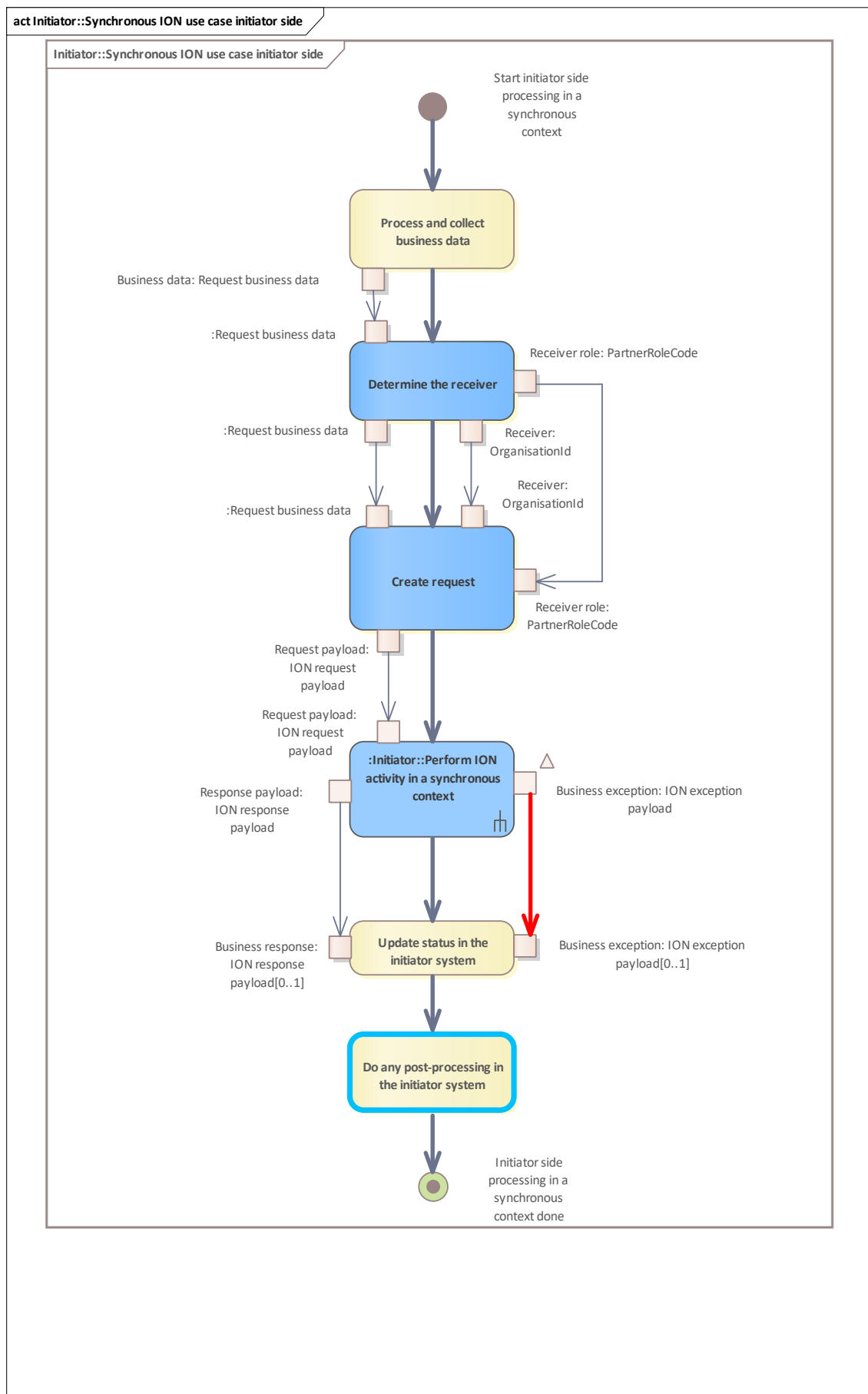
Linked Use Cases (Includes)	Receive and validate synchronous ION response / Compile and send ION request / Synchronous ION use case processor side
Linked Use Cases (Realises)	
Base Activity	
Inputs	Request payload : ION request payload
Outputs	Response payload : ION response payload
Error Cases	Business exception : ION exception payload
Activity Diagram	Initiator::Perform ION activity in a synchronous context
Alternative 1	
Inputs	
Outputs	
Error Cases	
Activity Diagram	Initiator::Synchronous ION use case initiator side

6.1.2.1.1 Initiator::Synchronous ION use case initiator side

Activity that shows sample steps and checks on the initiator side in a synchronous context. This activity waits synchronously when the initiator-side business logic has finished and the message has been sent from the [Initiator](#) to the [Processor](#) and continues as soon as the ION response is returned by the [Processor](#).



6.1.2.1.1 Initiator::Synchronous ION use case initiator side



Activity diagram which shows the control flow, object flow and potential exception flow in the activity [Sample process client side in synchronous context](#) triggered by a [Synchronous use case client side](#).

Process and collect business data

Abstract action that stands for the concrete business logic which collects the necessary data in one or more steps.

If no business data is needed (e.g. for pure get-scenarios), this step will only indicate which data is needed from a third-party system (getXXX).

Determine the receiver

E.g. in notification scenarios, the receiver organisation can be extracted from the contained binary transaction attestation. The receiver role comes from the use case context (e.g. a notification about an entitlement blocking is always initially sent to the PO) based on the information of the intended called operation that was called.

Note: One organisation may have more than one role.

Create request

Create the request as an XSD top level element containing the [Request business data](#), together with the [CommunicationInformation](#) as far as possible. Do not consider the creation of the [IonMessageId](#) or [ProcessInstanceId](#). This is done later during the ION message compilation. The own organisation ID, as well as the own role, is also added later in the ION message compilation, see [Compile and send ION request](#).

The receiver role (and service) has to be set only if the use case does not determine it.

Initiator::Perform ION activity in a synchronous context

See [Initiator::Perform ION activity in a synchronous context](#).

Update status in the initiator system

A business response or exception is received and status is updated accordingly in the initiator's system.

The generic message correlation steps are done before in the use cases [Receive and validate asynchronous ION response](#), [Receive and validate asynchronous business exception](#), [Receive and validate synchronous ION response](#) or [Receive and validate synchronous ION business exception](#).

This step should update the status of a certain entity depending on the underlying use case (e.g. set the state of an entitlement to "hotlisted").

In the case of a business exception: depending on exception type, a subsequent handling may be required after updating the status to enable regular processing in further actions. **It is assumed that the problem defined in the exception is solved before the action completes. Only then, the follow-up steps can continue.**

In the case of warnings contained in the reply, a further warning handling could be necessary. If the [response payload](#) contains [Response business data](#), this business data must be processed in further steps.

Do any post-processing in the initiator system

Do post-processing depending on the underlying use case.

If the [Response payload](#) contains [Response business data](#), this business data must be processed here.

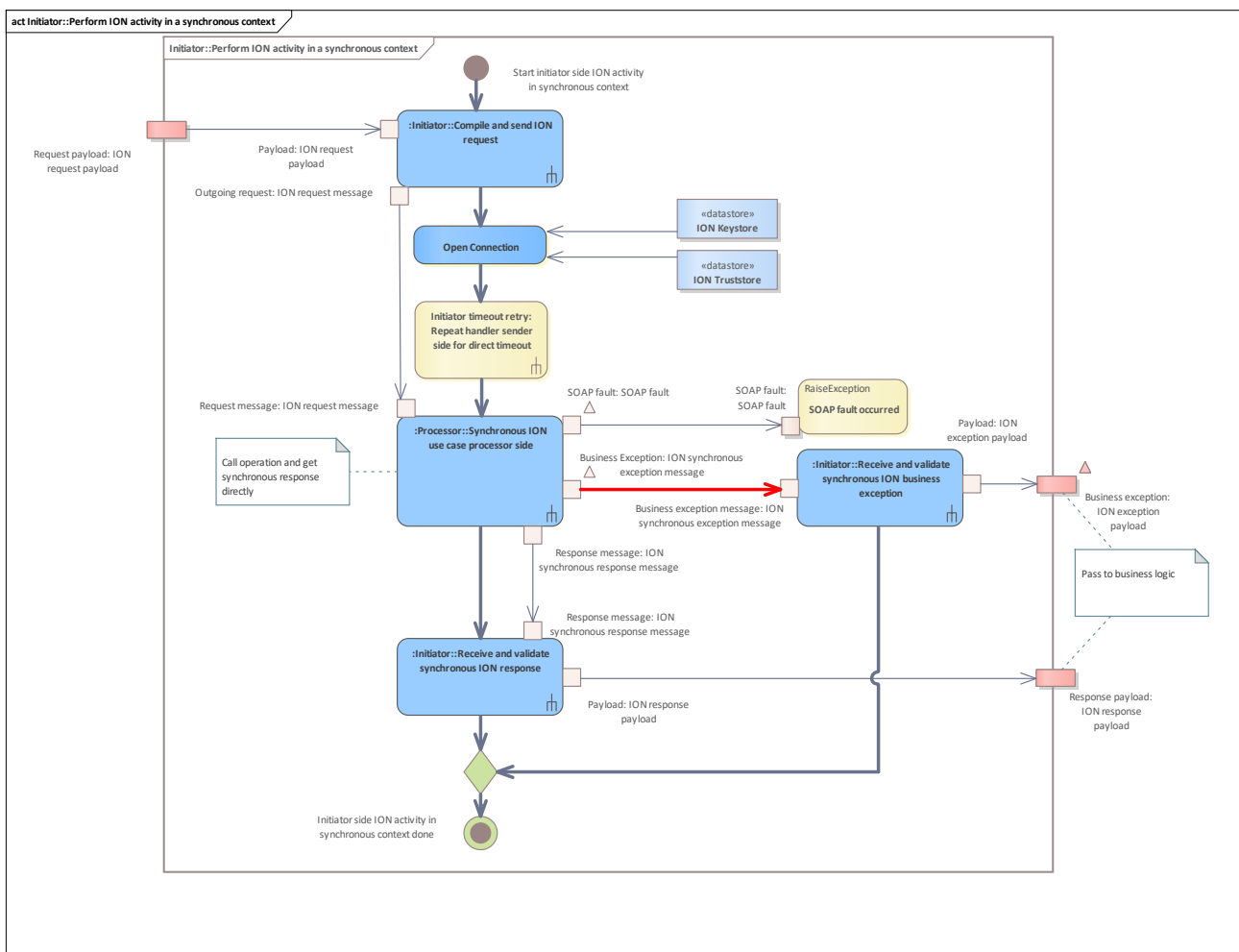
Any other post-processing is also possible here. This might be something out of scope like sub-ledger steps or something specified in etiCORE like further ION calls or calling an included use case.

6.1.2.1.1.2 Initiator::Perform ION activity in a synchronous context

Activity that shows the common steps and checks to be performed if the ION communication on the initiator side takes place in a synchronous context. This activity starts when the initiator-side business logic has finished and the message has to be sent from the [Initiator](#) to the [Processor](#).

The underlying or calling process [Sample process initiator side in a synchronous context](#) becomes active again if the ION response occurs and its business data is forwarded there.

6.1.2.1.1.2.1 Initiator::Perform ION activity in a synchronous context



Activity diagram which shows the control flow, object flow and potential exception flow in the activity [Process initiator side ION activity in synchronous context](#) triggered by the [Synchronous use case initiator side](#).

Initiator::Compile and send ION request

See [Initiator::Compile and send ION request](#).

Processor::Synchronous ION use case processor side

See [Processor::Synchronous ION use case processor side](#).

Initiator::Receive and validate synchronous ION response

See [Initiator::Receive and validate synchronous ION response](#).

Initiator::Receive and validate synchronous ION business exception

See [Initiator::Receive and validate synchronous ION business exception](#).

Repeat handler sender side for direct timeout

See [Repeat handler sender side for direct timeout](#).

SOAP fault occurred

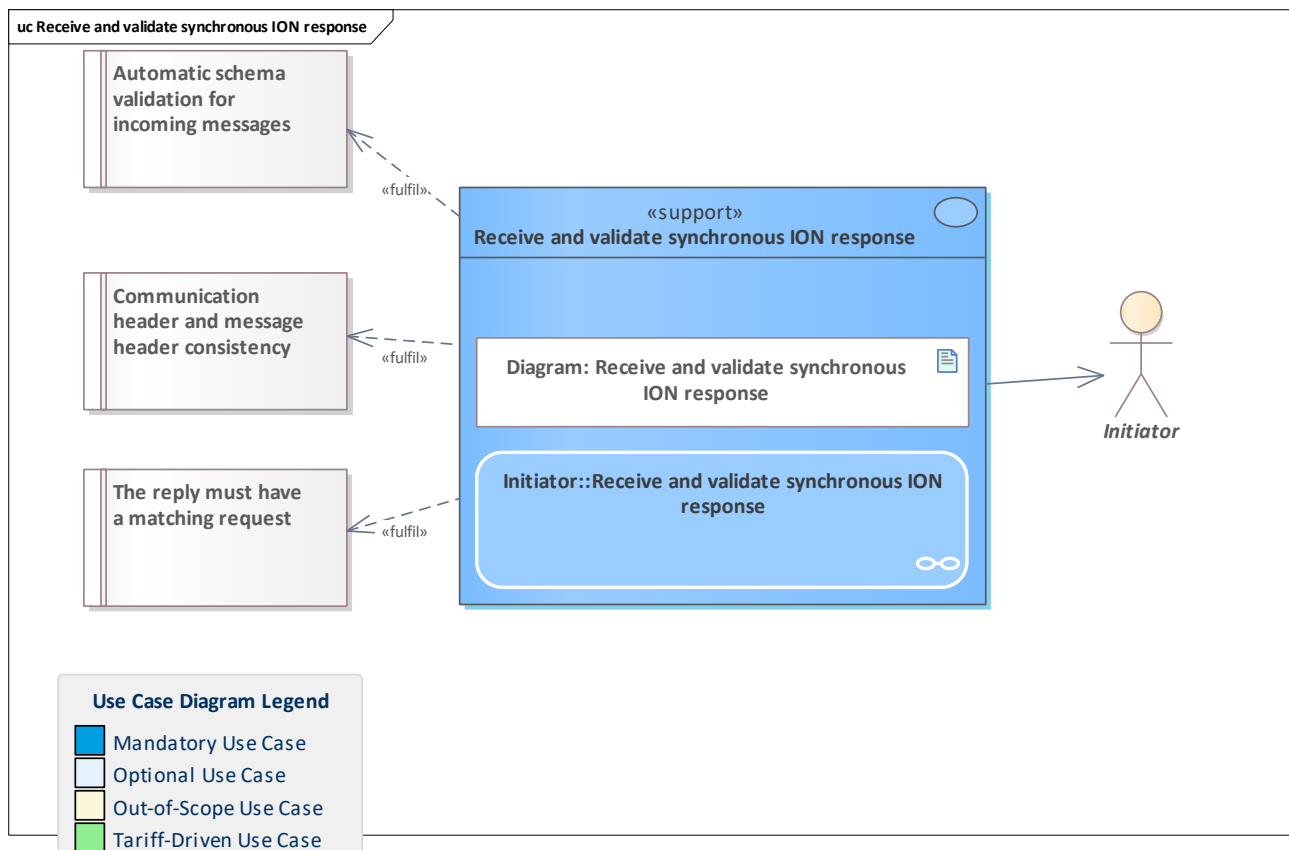
Handle a [SOAP fault](#). Normally, the underlying process will be aborted.

Open Connection

Opens a TLS connection (to the CRE) presenting the [ION TLS certificate](#) . In case the CRE URL uses the HTTP scheme, open TCP connection or re-use an existing one.

Opens a TLS connection to the CRE presenting the [ION TLS certificate](#) found in the [ION Keystore](#) (see also [Keys and certificates : ION Key- and Truststore](#)) for the configured organisation ID. Checks the validity of the retrieved certificate against the [ION Truststore](#) (incorporating CRLs or OCSP). Asserts that the organisation ID given in the "o" part of the certificate subject matches 5602 (organisation ID of the CRE) and that there is no "ou" part (i.e. it is an TLS certificate).

6.1.2.2 Receive and validate synchronous ION response



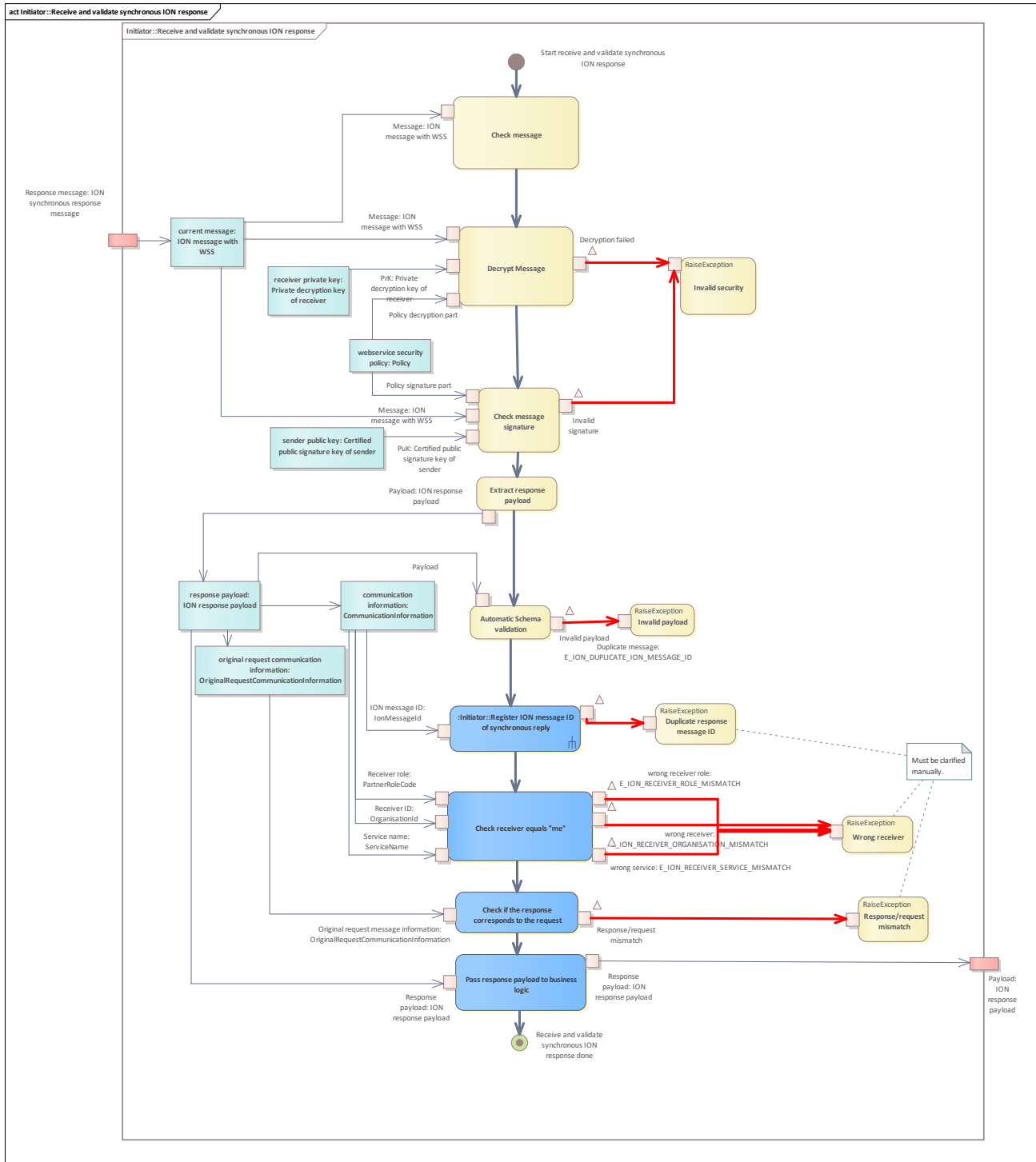
Use Case	Receive and validate synchronous ION response
Description	<p>The general use case for receiving a synchronous Processor's response message by the Initiator.</p> <p>The received synchronous ION response message was sent directly and synchronously within the same operation. It contains a response payload which is a suitable use case-specific top-level element and named after the operation provided by the Processor which was called by the Initiator, extended by the keyword "Response". For example, addXXToHotlist will be addXXToHotlistResponse.</p> <p>The element contains either a BusinessAcknowledgement only or a combined response with further Response business data coming from the processor.</p> <p>The Initiator has to ensure that the message is authentic and that the syntax and content of the response are valid.</p> <p>This scenario is valid for all derived use cases which have to receive synchronous responses.</p> <p>For the BusinessAcknowledgement part of the response, the main step is to find the reference of the original message by the delivered reference.</p> <p>If the reference cannot be found, no external exception is generated, since the parent use case ends with this acknowledgement.</p> <p>This problem has to be treated internally or with the processor's notifyEvent method.</p> <p>This is also the case if the optional additional Response business data causes any problems in the initiator's system.</p>
Initiating Actor	
Reacting Actor	Initiator

Preconditions	
Postconditions	
Linked Use Cases (Extended By)	
Linked Use Cases (Includes)	
Linked Use Cases (Realises)	
Base Activity	
Inputs	Response message : ION synchronous response message
Outputs	Payload : ION response payload
Error Cases	
Activity Diagram	Initiator::Receive and validate synchronous ION response

6.1.2.2.1 Initiator::Receive and validate synchronous ION response

See [Receive and validate synchronous ION response](#).

6.1.2.2.1.1 Initiator::Receive and validate synchronous ION response



Activity diagram which shows the control flow, object flow and potential exception flow triggered by the use case [Receive and validate synchronous ION response](#).

Check message

Check if the SOAP message is well-formed and complies with the standard SOAP schema and the binding defined in the WSDL:

- SOAP message in general
- SOAP header composition

- WS-policy

Normally performed by the underlying framework.

Pass response payload to business logic

Allow the response - represented by the response payload - to be returned to the higher-level business logic after the generic checks have been performed.

Extract response payload

Normally performed by the underlying web service framework. The payload is the etiCORE defined embedded top-level element in the [SOAP Body](#) always as inherited [BusinessAcknowledgement](#) and, optionally, implements [Response business data](#) (here: abstract placeholder for all implementing business data response classes).

Invalid payload

Exception if an automatic schema validation error occurs.

Two scenarios may occur:

- If the automatic schema validation of the incoming message fails, the software implementation of the sender has to be fixed. The called operation terminates with this exception and no processing takes place
- If the automatic schema validation of the outgoing message fails, your software implementation or data storage has to be fixed. The process stops with this exception and prevents the case from above

For incoming messages, the underlying framework has to respect the requirement [Security relevant errors](#) to avoid DoS attacks. If the framework does not fulfil it, a separate exception handler must be employed.

Invalid security

Exception if the message cannot be decrypted or that the message's signature cannot be verified.

The most commonly used error message in the different WSS frameworks is "Invalid security header".

In most of these cases, a wrong key reference is sent, the normalisation of the message is wrong or the message does not meet the requirements of the etiCORE WS policy (see [Appendix: etiCORE Standard-Policy](#)).

The underlying framework has to respect the requirement [Security relevant errors](#). If the framework does not fulfil it, a separate exception handler must be employed.

Duplicate response message ID

Potential runtime exception if the response message ID is already in the system. In the synchronous context, a business exception is not possible.

Response/request mismatch

Potential runtime exception if the correlation in the response's [OriginalRequestCommunicationInformation](#) does not match the current request message ID ([IonMessageId](#)). In the synchronous context, a business exception is not possible.

Check if the response corresponds to the request

In synchronous context, assert the equality of the [OriginalRequestCommunicationInformation](#)'s copied original request [IonMessageId](#) and timestamp in the response with the corresponding fields in the [CommunicationInformation](#) of the current request.

Automatic Schema validation

Do the automatic XML schema validation with the embedded payload.
Normally performed by the underlying web service framework.

Check message signature

Check the compliance to the WSS policy described in the chapter [Message-based encryption \(application layer\)](#) and [Appendix: etiCORE Standard-Policy](#).

Check the signature with the right certified public key from the directory service of the PKI. Identify the certificate matching all of the following parameters

- the organisation ID of the sender (with certificate subject)
- the role of the sender (with certificate subject)
- the purpose ("sig"), see [Basic Requirements](#)
- the serial number of the key located in the security header

Furthermore, consider the rules concerning the [Validity of Certificates](#).

Check that the signature was performed by the sender and its role is located in the SOAP Header.

Partially performed by the underlying web service framework's security provider.

Note: due to the etiCORE [Policy](#), only the signature key's serial number is contained in the [Security header](#) of the [SOAP header](#) in clear text.

Note: consider the [Transition Phase for Certificate Renewal](#) and [Security Levels](#).

Decrypt Message

Check the compliance to the WSS policy described in the chapter [Message-based encryption \(application layer\)](#) and [Appendix: etiCORE Standard-Policy](#).

Decrypt the incoming SOAP message using your own private key for

- the organisation
- the role
- the purpose (here: decryption)

Only the SOAP body is encrypted. The encryption information is located in the security header embedded in the SOAP header.

Normally performed by the underlying web service framework's security provider.

Note: consider the [Transition Phase for Certificate Renewal](#) and [Security Levels](#).

Check receiver equals "me"

The receiver of the message must be my organisation ID and my role (see [PartnerRoleEnum](#)) and my service.

This check for organisation and role will never fail as long as the WSS is switched on and only becomes relevant without WSS.

The check for the service name may fail even if WSS is switched on since the service name is not part of the WSS parameters. A service name mismatch may only occur if a partner configured a wrong URL for its service.

Initiator::Register ION message ID of synchronous reply

See [Initiator::Register ION message ID of synchronous reply](#).

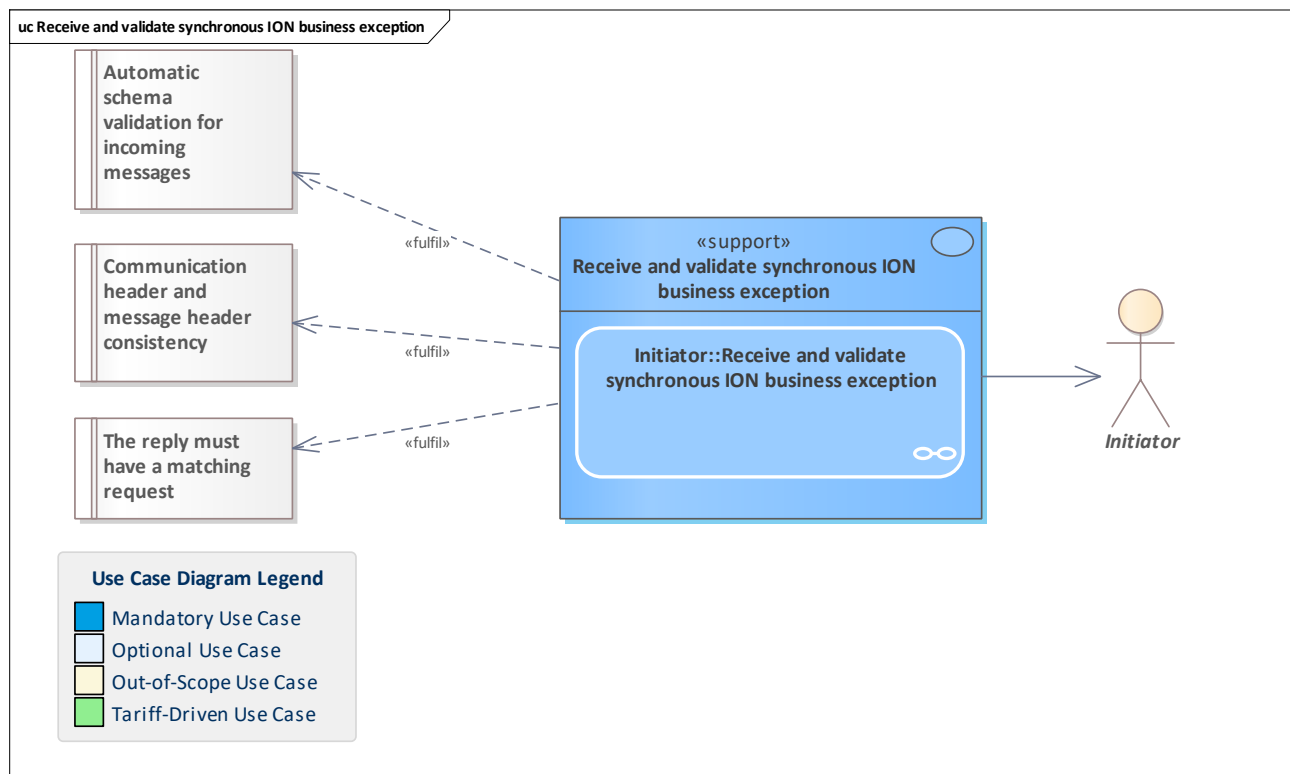
Wrong receiver

Potential runtime exception if the intended receiver is not me. In the synchronous context, a business exception is not possible.

6.1.2.2 Diagram: Receive and validate synchronous ION response

A diagram that shows the supporting use case done by the [Initiator](#) to receive and validate a synchronous ION response. This use case contains all generic steps and checks to be done before the message is passed to the [Initiator](#) system's business logic. Furthermore, the diagram shows the requirements to be fulfilled by the use case.

6.1.2.3 Receive and validate synchronous ION business exception



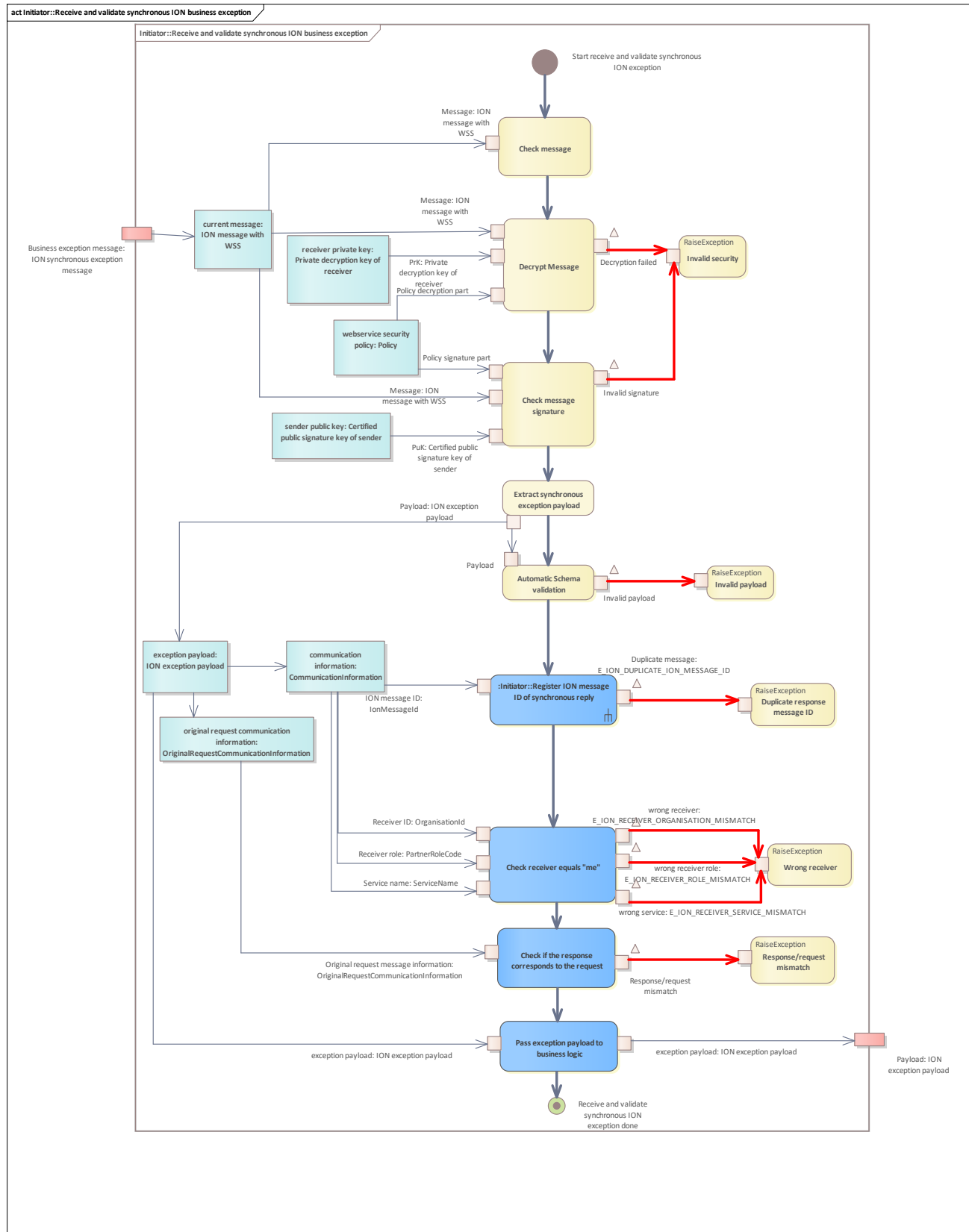
Use Case	Receive and validate synchronous ION business exception
Description	<p>Common use case to receive a BusinessException synchronously by the Initiator.</p> <p>The provided message synchronous ION exception message is sent directly within the same operation by the Processor.</p> <p>This use case is only used in the synchronous context.</p>

	<p>This scenario is valid for all derived use cases which have to receive synchronous business exceptions.</p> <p>The delivered exception has an additional Error to be evaluated and possibly handled afterwards, as well as possible events from the contained WarningList.</p> <p>Since no further business data is contained, the main step is to find the reference of the original message which can be done directly in the synchronous context. This reference is delivered in the business exception.</p> <p>If the reference cannot be found, no external exception is generated, since the parent use case ends with this acknowledgement. This problem has to be treated internally or by using the processor's notifyEvent method.</p>
Initiating Actor	
Reacting Actor	Initiator
Preconditions	
Postconditions	
Linked Use Cases (Extended By)	
Linked Use Cases (Includes)	
Linked Use Cases (Realises)	
Base Activity	
Inputs	Business exception message : ION synchronous exception message
Outputs	Payload : ION exception payload
Error Cases	
Activity Diagram	Initiator::Receive and validate synchronous ION business exception

6.1.2.3.1 Initiator::Receive and validate synchronous ION business exception

See [Receive and validate synchronous ION exception](#).

6.1.2.3.1.1 Initiator::Receive and validate synchronous ION business exception



Activity diagram for [Receive and validate synchronous ION exception](#).

Decrypt Message

Check the compliance to the WSS policy described in the chapter [Message-based encryption \(application layer\)](#) and [Appendix: etiCORE Standard-Policy](#).

Decrypt the incoming SOAP message using your own private key for

- the organisation
- the role
- the purpose (here: decryption)

Only the SOAP body is encrypted. The encryption information is located in the security header embedded in the SOAP header.

Normally performed by the underlying web service framework's security provider.

Note: consider the [Transition Phase for Certificate Renewal](#) and [Security Levels](#).

Check message signature

Check the compliance to the WSS policy described in the chapter [Message-based encryption \(application layer\)](#) and [Appendix: etiCORE Standard-Policy](#).

Check the signature with the right certified public key from the directory service of the PKI.

Identify the certificate matching all of the following parameters

- the organisation ID of the sender (with certificate subject)
- the role of the sender (with certificate subject)
- the purpose ("sig"), see [Basic Requirements](#)
- the serial number of the key located in the security header

Furthermore, consider the rules concerning the [Validity of Certificates](#).

Check that the signature was performed by the sender and its role is located in the SOAP Header.

Partially performed by the underlying web service framework's security provider.

Note: due to the etiCORE [Policy](#), only the signature key's serial number is contained in the [Security header](#) of the [SOAP header](#) in clear text.

Note: consider the [Transition Phase for Certificate Renewal](#) and [Security Levels](#).

Invalid security

Exception if the message cannot be decrypted or that the message's signature cannot be verified.

The most commonly used error message in the different WSS frameworks is "Invalid security header".

In most of these cases, a wrong key reference is sent, the normalisation of the message is wrong or the message does not meet the requirements of the etiCORE WS policy (see [Appendix: etiCORE Standard-Policy](#)).

The underlying framework has to respect the requirement [Security relevant errors](#). If the framework does not fulfil it, a separate exception handler must be employed.

Extract synchronous exception payload

Extract the exception payload element (type [BusinessException](#)) from the wrapped [SOAP Fault](#).

Automatic Schema validation

Do the automatic XML schema validation with the embedded payload.
Normally performed by the underlying web service framework.

Check if the response corresponds to the request

In synchronous context, assert the equality of the [OriginalRequestCommunicationInformation](#)'s copied original request [IonMessageId](#) and timestamp in the response with the corresponding fields in the [CommunicationInformation](#) of the current request.

Response/request mismatch

Potential runtime exception if the correlation in the response's [OriginalRequestCommunicationInformation](#) does not match the current request message ID ([IonMessageId](#)). In the synchronous context, a business exception is not possible.

Pass exception payload to business logic

Allow the exception - represented by the exception payload - to be returned to the higher- level business logic after the generic checks failed.

Invalid payload

Exception if an automatic schema validation error occurs.

Two scenarios may occur:

- If the automatic schema validation of the incoming message fails, the software implementation of the sender has to be fixed. The called operation terminates with this exception and no processing takes place
- If the automatic schema validation of the outgoing message fails, your software implementation or data storage has to be fixed. The process stops with this exception and prevents the case from above

For incoming messages, the underlying framework has to respect the requirement [Security relevant errors](#) to avoid DoS attacks. If the framework does not fulfil it, a separate exception handler must be employed.

Duplicate response message ID

Potential runtime exception if the response message ID is already in the system. In the synchronous context, a business exception is not possible.

Check message

Check if the SOAP message is well-formed and complies with the standard SOAP schema and the binding defined in the WSDL:

- SOAP message in general
- SOAP header composition
- WS-policy

Normally performed by the underlying framework.

Check receiver equals "me"

The receiver of the message must be my organisation ID and my role (see [PartnerRoleEnum](#)) and my service.

This check for organisation and role will never fail as long as the WSS is switched on and only becomes relevant without WSS.

The check for the service name may fail even if WSS is switched on since the service name is not part of the WSS parameters. A service name mismatch may only occur if a partner configured a wrong URL for its service.

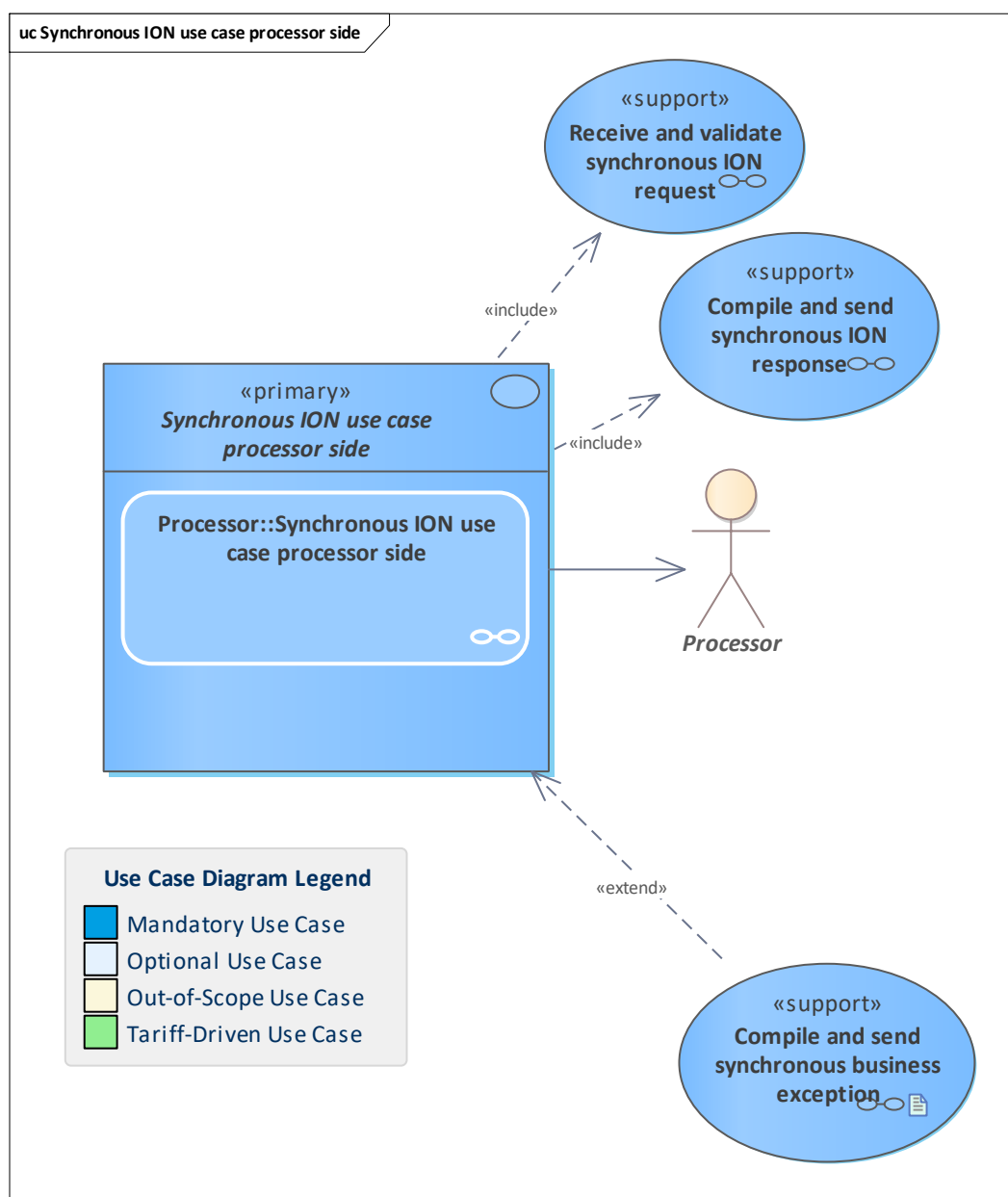
Initiator::Register ION message ID of synchronous reply

See [Initiator::Register ION message ID of synchronous reply](#).

Wrong receiver

Potential runtime exception if the intended receiver is not me. In the synchronous context, a business exception is not possible.

6.1.2.4 Synchronous ION use case processor side



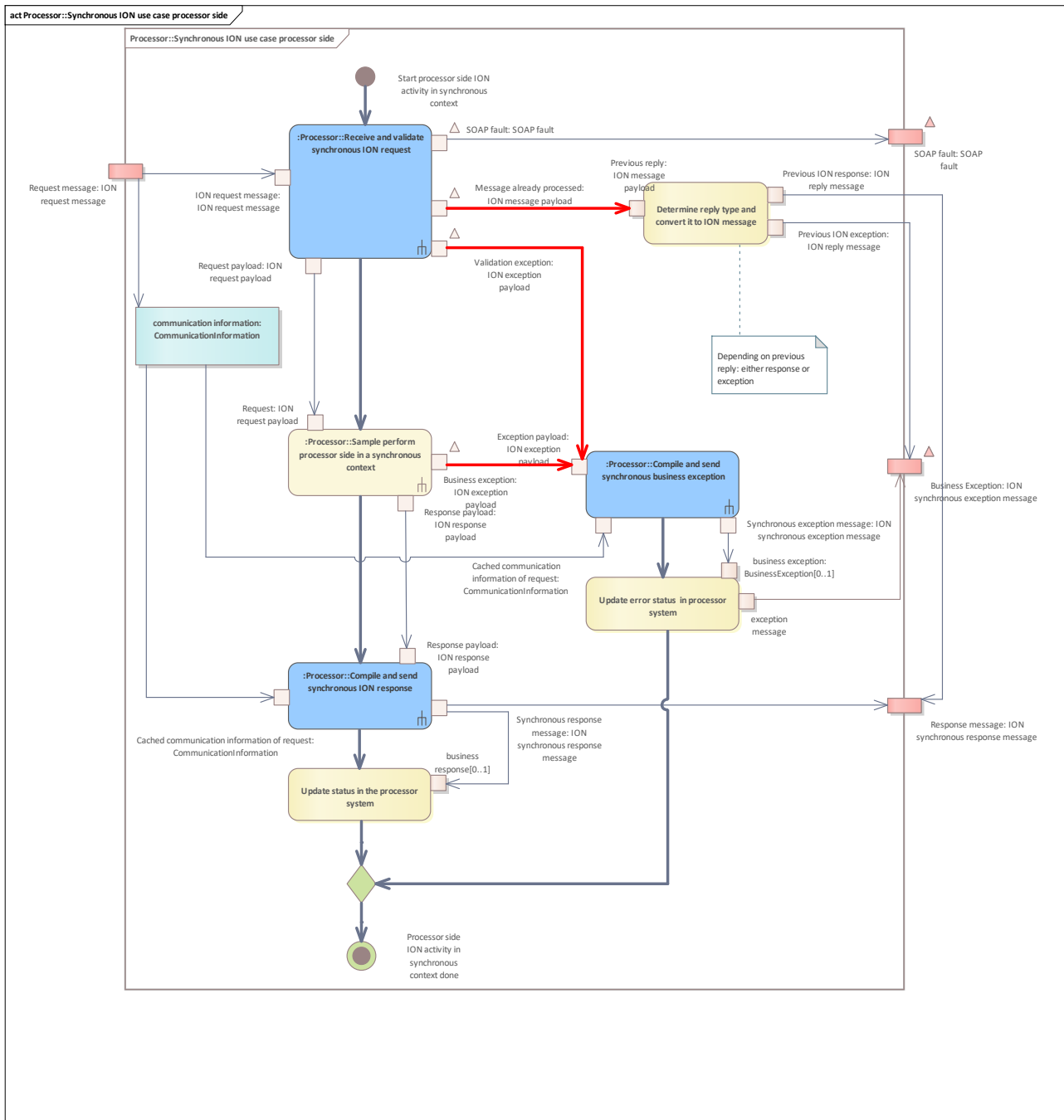
Use Case	Synchronous ION use case processor side
Description	<p>This use case covers the whole processor side in a synchronous use case context performed by the Processor.</p> <p>The steps are:</p> <ul style="list-style-type: none"> • Receive and validate ION request • Perform the business logic and, depending on the use case, collect business data for a response • Compile and send synchronous ION response
Initiating Actor	
Reacting Actor	Processor
Preconditions	
Postconditions	
Linked Use Cases (Extended By)	Compile and send synchronous business exception
Linked Use Cases (Includes)	Compile and send synchronous ION response / Receive and validate synchronous ION request
Linked Use Cases (Realises)	
Base Activity	
Inputs	Request message : ION request message
Outputs	Response message : ION synchronous response message
Error Cases	E ION RECEIVER ROLE MISMATCH E ION RECEIVER SERVICE MISMATCH E ION RECEIVER ORGANISATION MISMATCH E ION UNKNOWN SENDER E ION WRONG SENDER ROLE E ION WRONG SENDER SERVICE E ION SAME MESSAGE IN PROGRESS E ION DUPLICATE ION MESSAGE ID E ION DUPLICATE ION MESSAGE ID E ION DUPLICATE ION MESSAGE ID SOAP fault : SOAP fault Business Exception : ION synchronous exception message
Activity Diagram	Processor::Synchronous ION use case processor side

6.1.2.4.1 Processor::Synchronous ION use case processor side

Activity that shows the common steps and checks to be performed if the ION communication on processor side is involved in a synchronous context. This activity embeds the processor-side business logic [Sample process processor side in synchronous context](#) in the order

- [Receive and validate ION request](#)
- [Sample process processor side in synchronous context](#)
- [Compile and send synchronous ION response](#) or [Compile and send synchronous business exception](#)

6.1.2.4.1.1 Processor::Synchronous ION use case processor side



Activity diagram which shows the control flow, object flow and potential exception flow in the activity [Process processor side ION activity in synchronous context](#) triggered by the [Synchronous use case processor side](#).

Processor::Sample perform processor side in a synchronous context

See [Processor::Sample perform processor side in a synchronous context](#).

Processor::Compile and send synchronous ION response

See [Processor::Compile and send synchronous ION response](#).

Update status in the processor system

Update the status in the processor's system concerning the processed request and the response sent to the initiator. The final status update is the same for synchronous or asynchronous contexts.

In the case of an incoming [deliveryRejection](#), consider a manual scenario.

Processor::Compile and send synchronous business exception

See [Processor::Compile and send synchronous business exception](#).

Update error status in processor system

Update the error status registering the occurred error and the reference to the sent exception.

In the case of an incoming [deliveryRejection](#), consider a manual scenario.

Determine reply type and convert it to ION message

Action that rebuilds an ION message for an existing [message payload](#) of a reply. Depending on the message type, either an [asynchronous ION response message](#) or an [asynchronous ION exception message](#) will be built and passed back in order to be directly sent to the sender.

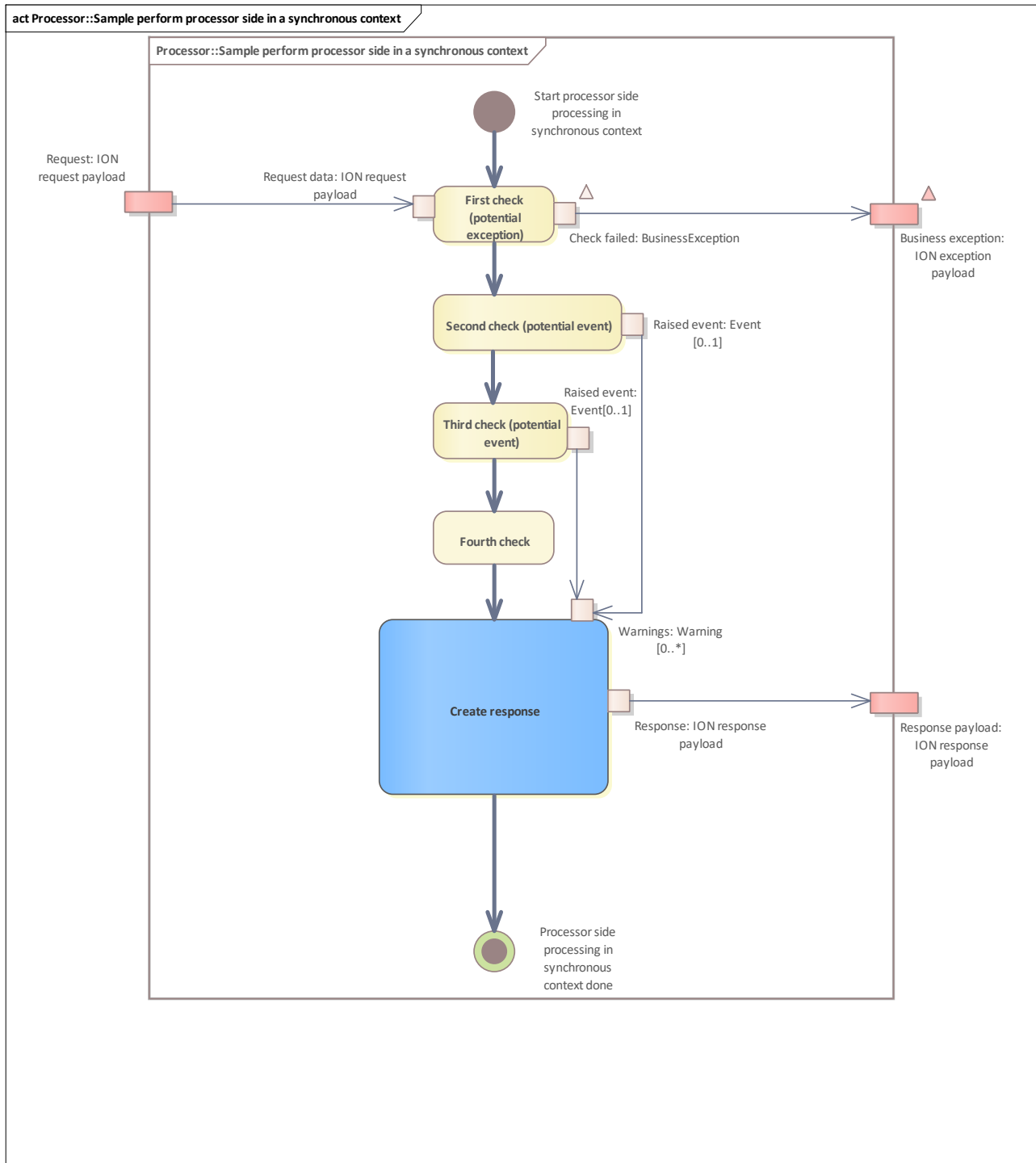
Processor::Receive and validate synchronous ION request

See [Processor::Receive and validate synchronous ION request](#).

6.1.2.4.1.2 Processor::Sample perform processor side in a synchronous context

Activity that shows sample steps and checks on the processor side in an asynchronous context. This activity is embedded between [Receive and validate ION request](#) and [Compile and send synchronous ION response](#) or [Compile and send synchronous business exception](#) and performs the processor-side business logic.

6.1.2.4.1.2.1 Processor::Sample perform processor side in a synchronous context



Activity diagram which shows the control flow, object flow and potential exception flow in the activity [Sample process processor side in synchronous context](#) triggered by the [Synchronous use case processor side](#).

First check (potential exception)

Example for a check which is done with the [Request business data](#). This example check may raise a [BusinessException](#).

Second check (potential event)

Example for a check which is done with the [Request business data](#). This example check may raise an [Event](#).

Third check (potential event)

Example for a check which is done with the [Request business data](#). This example check may raise a further [Event](#).

Fourth check

An example check that is not described in the etiCORE specification in a detailed way.

Create response

Template action to create a [Response payload](#) object.

Compile [Response business data](#) (if any beyond the [BusinessAcknowledgement](#)).

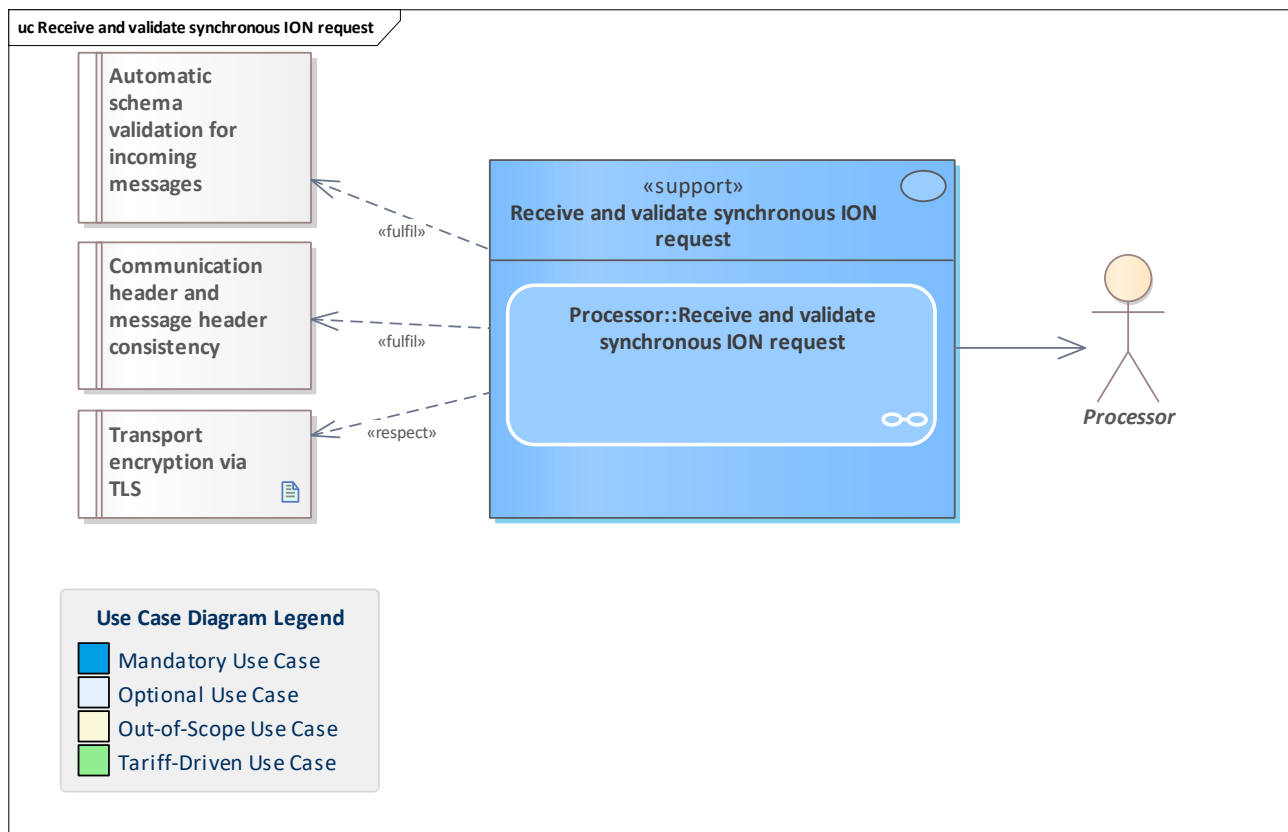
Build a top-level element for the response payload containing

- "empty" [OriginalRequestCommunicationInformation](#) and [CommunicationInformation](#) objects¹
- the [WarningList](#) with the processing events - if any
- the newly compiled [Response business data](#) - if any

¹To keep the same steps of filling the [OriginalRequestCommunicationInformation](#) and [CommunicationInformation](#) objects in the response payload away from the business logic, this is done later in [Processor::Compile and send an asynchronous ION response](#).

Also do not consider the (new) response ION message ID and your own organisation ID and role. These are also done there.

6.1.2.5 Receive and validate synchronous ION request



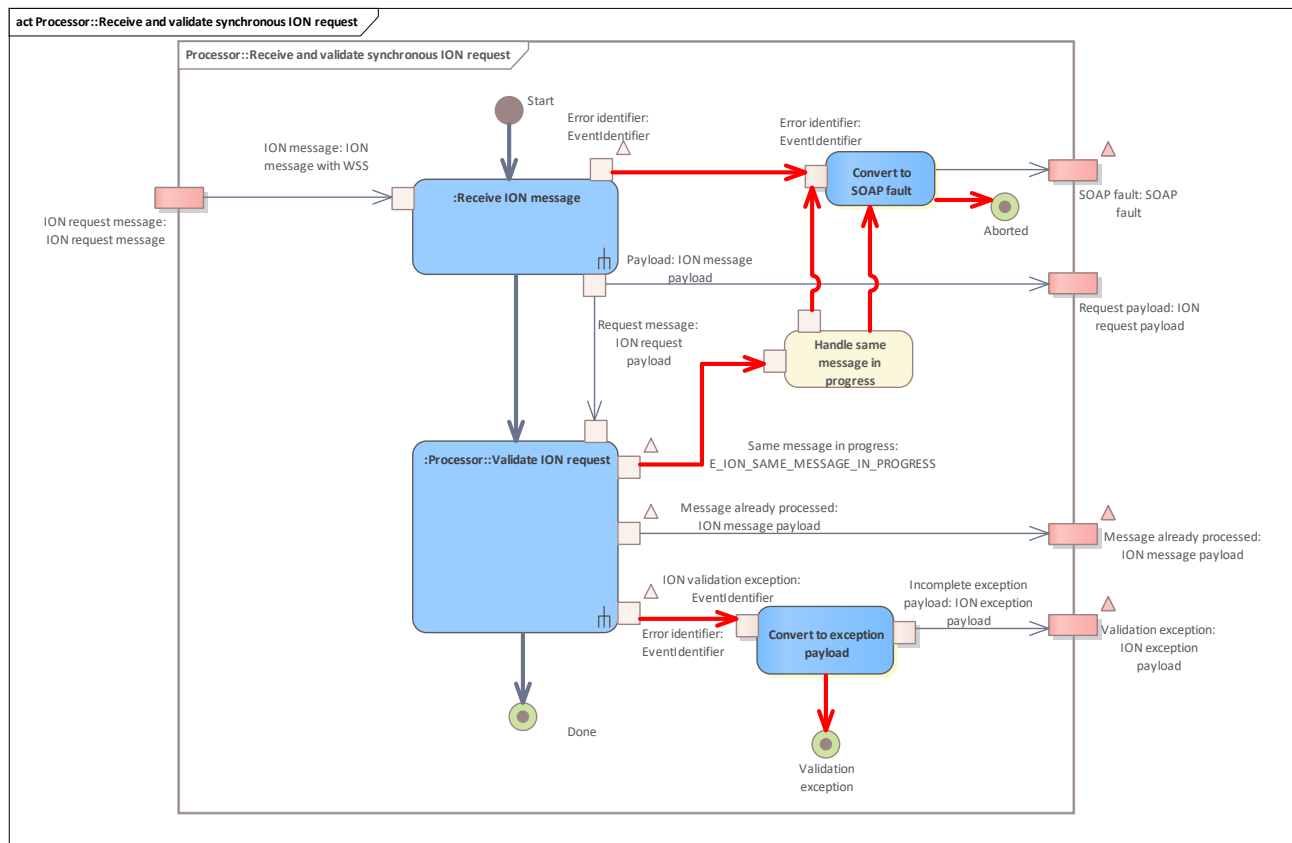
Use Case	Receive and validate synchronous ION request
Description	Use case that combines the activity Receive ION message and Processor::Validate ION request . The use case encapsulates all ION activities before the business part of the use case itself is started.
Initiating Actor	
Reacting Actor	Processor
Preconditions	
Postconditions	
Linked Use Cases (Extended By)	
Linked Use Cases (Includes)	
Linked Use Cases (Realises)	
Base Activity	
Inputs	Request payload : ION request payload ION request message : ION request message
Outputs	
Error Cases	E ION RECEIVER ROLE MISMATCH E ION RECEIVER SERVICE MISMATCH E ION RECEIVER ORGANISATION MISMATCH E ION UNKNOWN SENDER E ION WRONG SENDER ROLE E ION WRONG SENDER SERVICE E ION SAME MESSAGE IN PROGRESS E ION DUPLICATE ION MESSAGE ID E ION DUPLICATE ION MESSAGE ID

	E ION DUPLICATE ION MESSAGE ID Validation exception : ION exception payload Message already processed : ION message payload SOAP fault : SOAP fault
Activity Diagram	Processor::Receive and validate synchronous ION request

6.1.2.5.1 Processor::Receive and validate synchronous ION request

See [Receive and validate ION request in synchronous context](#).

6.1.2.5.1.1 Processor::Receive and validate synchronous ION request



See [Processor::Receive and validate ION request in synchronous context](#).

Receive ION message

See [Receive ION message](#).

Processor::Validate ION request

See [Processor::Validate ION request](#).

Convert to SOAP fault

Embed the incoming [EventIdentifier](#) into a [SOAP fault](#) with the specified format in [Asynchronous Reply Overview](#).

Handle same message in progress

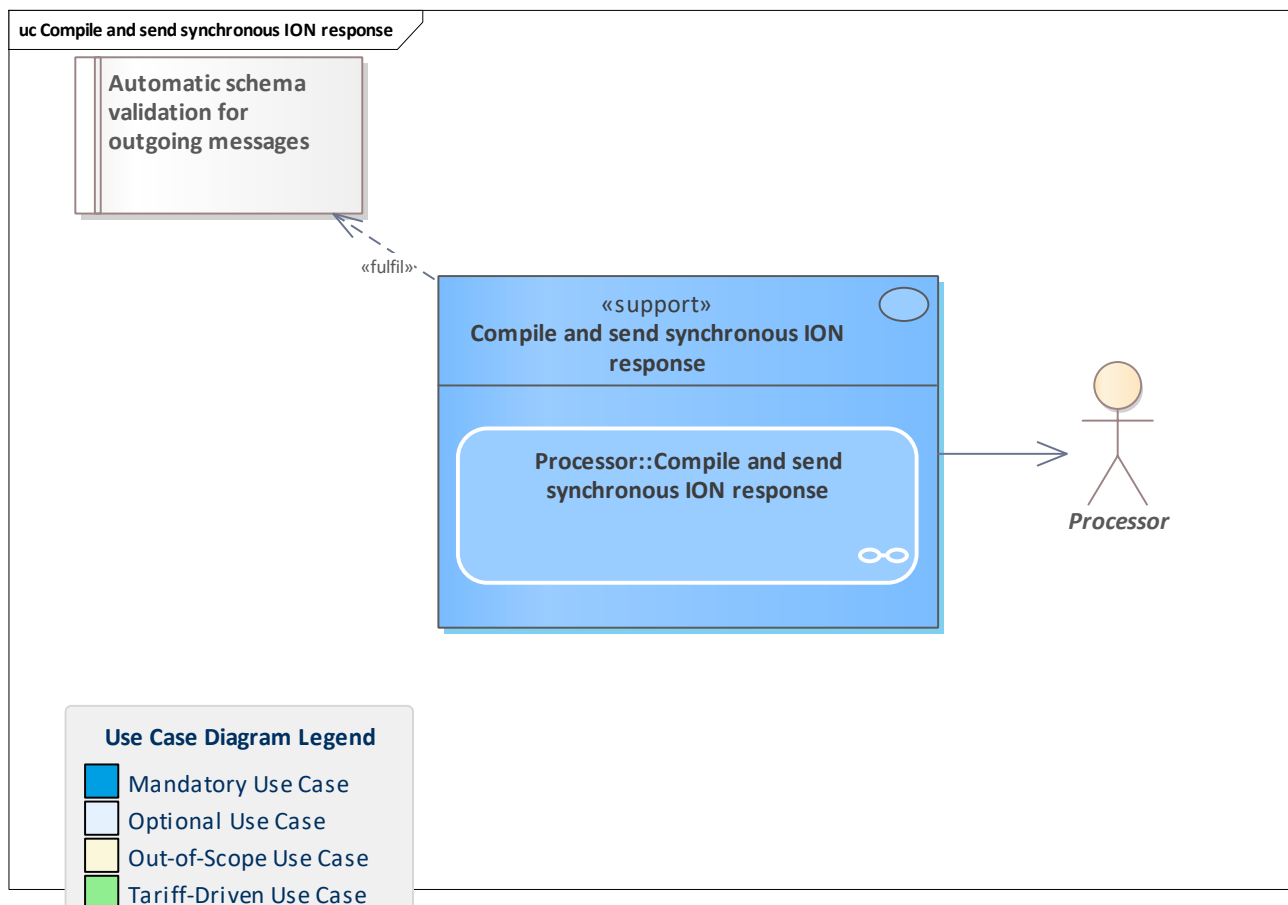
To keep the system's reaction simple due to possible system overload, just log and pass the [SOAP fault](#) caused by processing the same message to the [Initiator](#) and do not send a regular reply (in this case an exception).

Since the reply of the previous (same) message processing will be sent, this behaviour will not cause any problems on initiator's side.

Convert to exception payload

Take an incoming [EventIdentifier](#), create the matching exception element including the correlation information of the original request without certain fields ([IonMessageId](#), etc.) of the [CommunicationInformation](#) which have to be filled later when the exception is sent.

6.1.2.6 Compile and send synchronous ION response



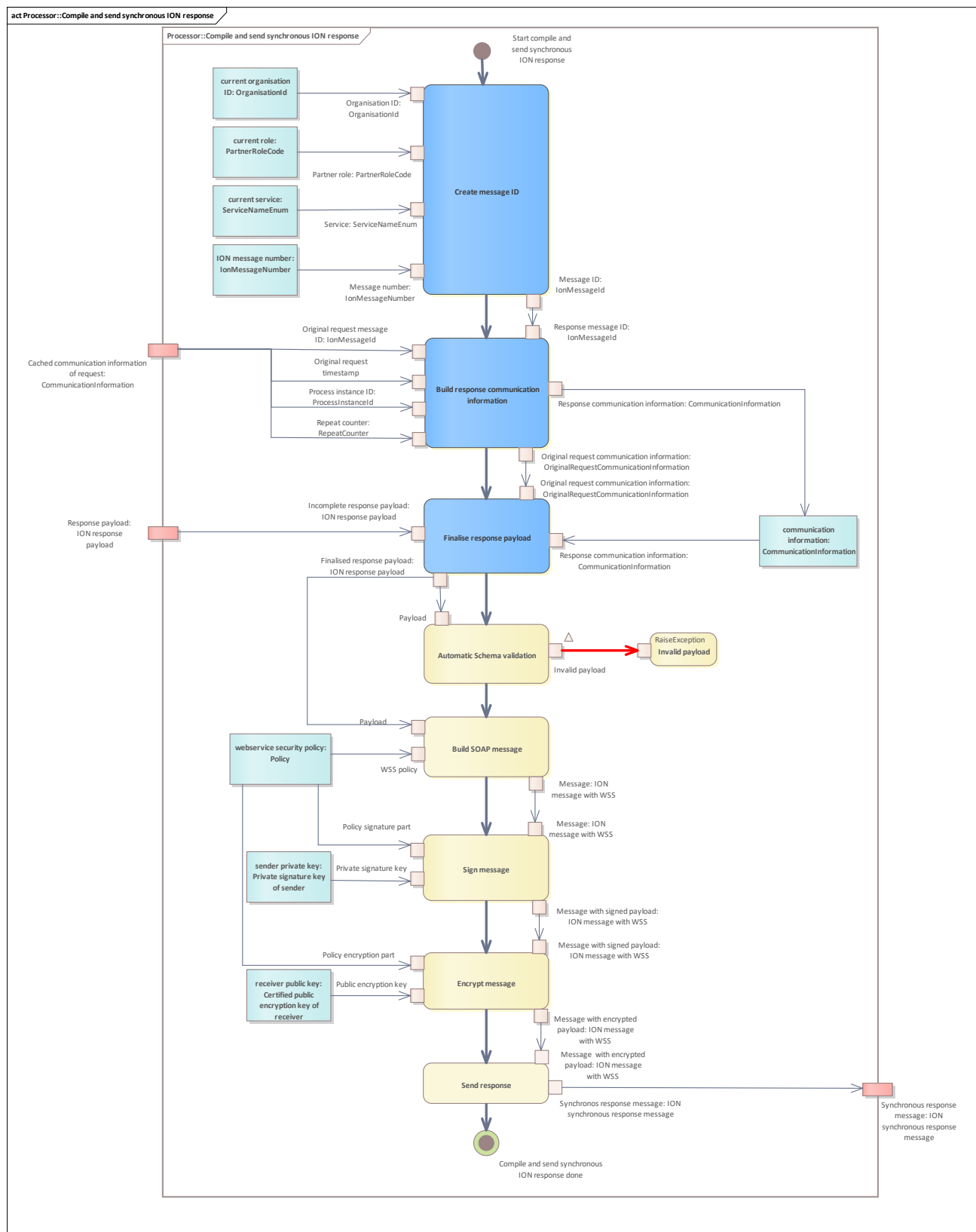
Use Case	Compile and send synchronous ION response
Description	<p>This supporting use case is performed by the Processor. The Processor has to ensure that the syntax and content of the message are valid (see Automatic schema validation for outgoing messages), that the message cannot be read by a third party (see ION:Privacy) and that the message is authentic (see ION:Authenticity).</p> <p>Compile and send a BusinessAcknowledgement type wrapped with a suitable use case-specific type related to a top-level element. This element is named after the operation provided by the</p>

	<p>Processor which was called by the Initiator extended by the keyword "Response". For example, addXXToHotlist will be addXXToHotlistResponse and will be sent directly and synchronously in the same operation, wrapped with a Synchronous ION response message.</p> <p>Depending on the underlying use case, events may occur in any step and are stored or collected in a WarningList.</p> <p>The further processing continues and is not stopped by an occurring Warning placed in the WarningList.</p> <p>The regular response is completed with a CommunicationInformation and a OriginalRequestCommunicationInformation object.</p>
Initiating Actor	
Reacting Actor	Processor
Preconditions	
Postconditions	
Linked Use Cases (Extended By)	
Linked Use Cases (Includes)	
Linked Use Cases (Realises)	
Base Activity	
Inputs	<p>Cached communication information of request : CommunicationInformation</p> <p>Response payload : ION response payload</p>
Outputs	Synchronous response message : ION synchronous response message
Error Cases	
Activity Diagram	Processor::Compile and send synchronous ION response

6.1.2.6.1 Processor::Compile and send synchronous ION response

Activity for the use case [Compile and send synchronous ION response](#).

6.1.2.6.1.1 Processor::Compile and send synchronous ION response



Activity diagram for the use case [Compile and send synchronous ION response](#) which shows the control flow, object flow and potential exception flow in the activity [Compile and send synchronous ION response](#) triggered by the use case [Synchronous use case processor side](#).

Build SOAP message

Build the SOAP message employing a [SOAP Envelope](#) with consideration of the binding rules in the corresponding WSDL. These rules determine if an [IonRoutingHeader](#) has to be embedded in the [SOAP Header](#).

Furthermore, the embedded webservice security policy in the WSDL is used to sign and encrypt the necessary parts of the message.

Encrypt message

Encrypt the message with the receiver's public key. This public key is certified and can be obtained from the PKI by its directory service.

Note: please consider the rules concerning the [Validity of Certificates](#).

Due to the web service security [Policy](#), the entire [SOAP Body](#) has to be encrypted. The [SOAP Header](#) remains in clear text for routing purposes. Furthermore, the signature of the [SOAP Body](#) also has to be encrypted. The encrypted signature data of the signed [SOAP Body](#) is stored in the [Security header](#) in the [SOAP Header](#) and is not part of the [SOAP Body](#).

Sign message

Sign the message parts as defined in the [Webservice Security Policy](#) with the private key for signature purposes (the corresponding public key is registered and certified in the PKI and can be accessed via the directory service of the PKI).

According to the web service security specification, the entire body of the SOAP message which contains the payload is signed.

Normally performed by the underlying WSS framework.

Automatic Schema validation

Do the automatic XML schema validation with the embedded payload.

Normally performed by the underlying web service framework.

Create message ID

Create a new [IonMessageId](#) which is embedded in the communication information of the payload.

Build response communication information

Complete the "existing" payload's communication information.

To build the [CommunicationInformation](#) object of the response:

- sender ID from request [IonMessageId](#) -> response receiver ID
- sender role from request [IonMessageId](#) -> response receiver role
- sender service from request [IonMessageId](#) -> response receiver service
- new [IonMessageId](#) for the response
- new message timestamp
- no [RepeatCounter](#)!
- request [ProcessInstanceId](#) -> response process instance ID

Take the following fields from [CommunicationInformation](#) of the original request and copy them to the response payload to build the [OriginalRequestCommunicationInformation](#):

- [IonMessageId](#)

- Timestamp
- [RepeatCounter](#) (if present)

Finalise response payload

Complete the "incomplete" response payload by adding the [OriginalRequestCommunicationInformation](#) object and the newly built [CommunicationInformation](#) object.

Invalid payload

Exception if an automatic schema validation error occurs.

Two scenarios may occur:

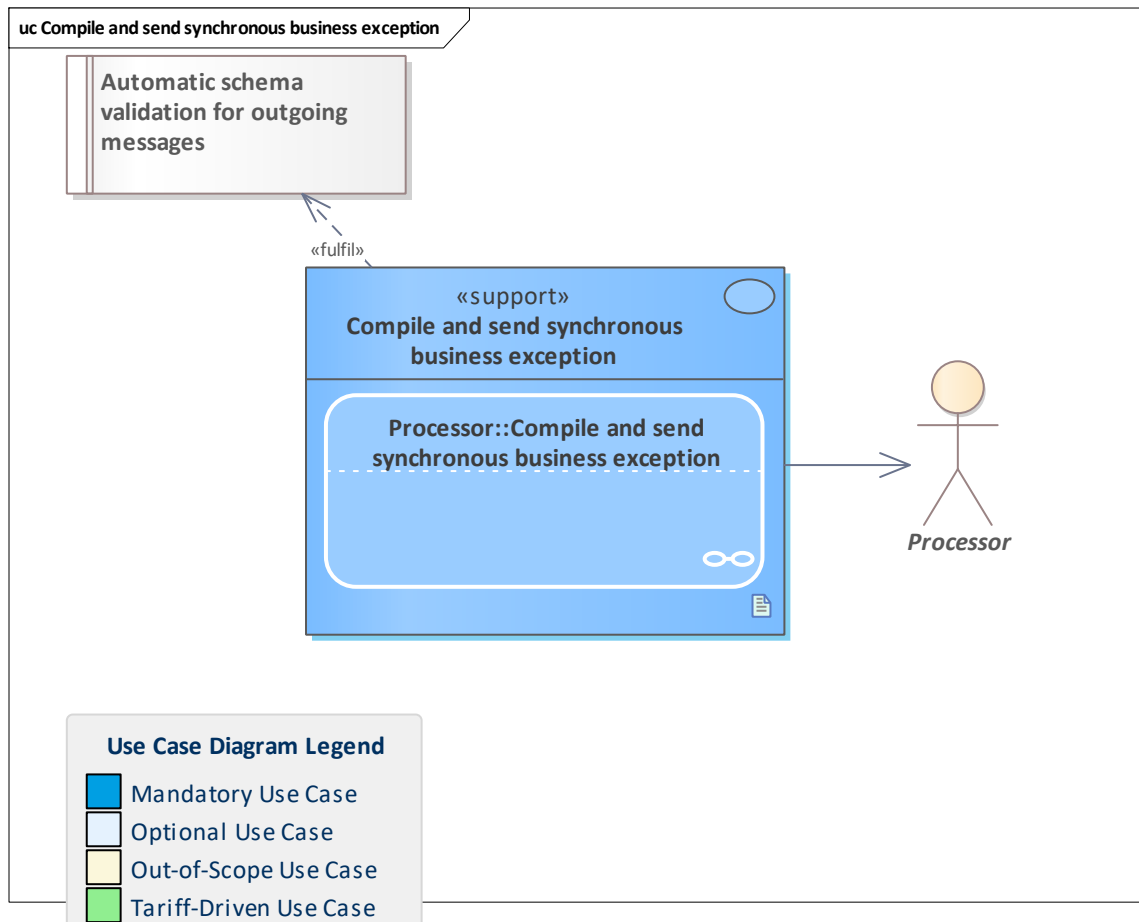
- If the automatic schema validation of the incoming message fails, the software implementation of the sender has to be fixed. The called operation terminates with this exception and no processing takes place
- If the automatic schema validation of the outgoing message fails, your software implementation or data storage has to be fixed. The process stops with this exception and prevents the case from above

For incoming messages, the underlying framework has to respect the requirement [Security relevant errors](#) to avoid DoS attacks. If the framework does not fulfil it, a separate exception handler must be employed.

Send response

Send the [Synchronous ION response message](#) with the signed and encrypted [Response payload](#).

6.1.2.7 Compile and send synchronous business exception



Use Case	Compile and send synchronous business exception
Description	<p>Compiles and sends a BusinessException named with a suitable use case-specific top-level element. This element is named after the operation provided by the Processor which was called by the Initiator and extended by the keyword "Exception". For example, addXXToHotlist will be addXXToHotlistException and will be sent directly and synchronously in the same operation, wrapped with a Synchronous ION exception message.</p> <p>The BusinessException transports the same information as a BusinessAcknowledgement, with an additional Error to be evaluated. This error may occur in any step.</p> <p>The further processing - apart from the fact that the business process does not continue - in the direction of the ION is no different.</p> <p>Instead of the regular response, the exception is completed with CommunicationInformation and wrapped in top-level element, SOAP Body and SOAP message with exactly the same steps.</p>
Initiating Actor	
Reacting Actor	Processor
Preconditions	
Postconditions	
Linked Use Cases (Extended By)	
Linked Use Cases (Includes)	
Linked Use Cases (Realises)	



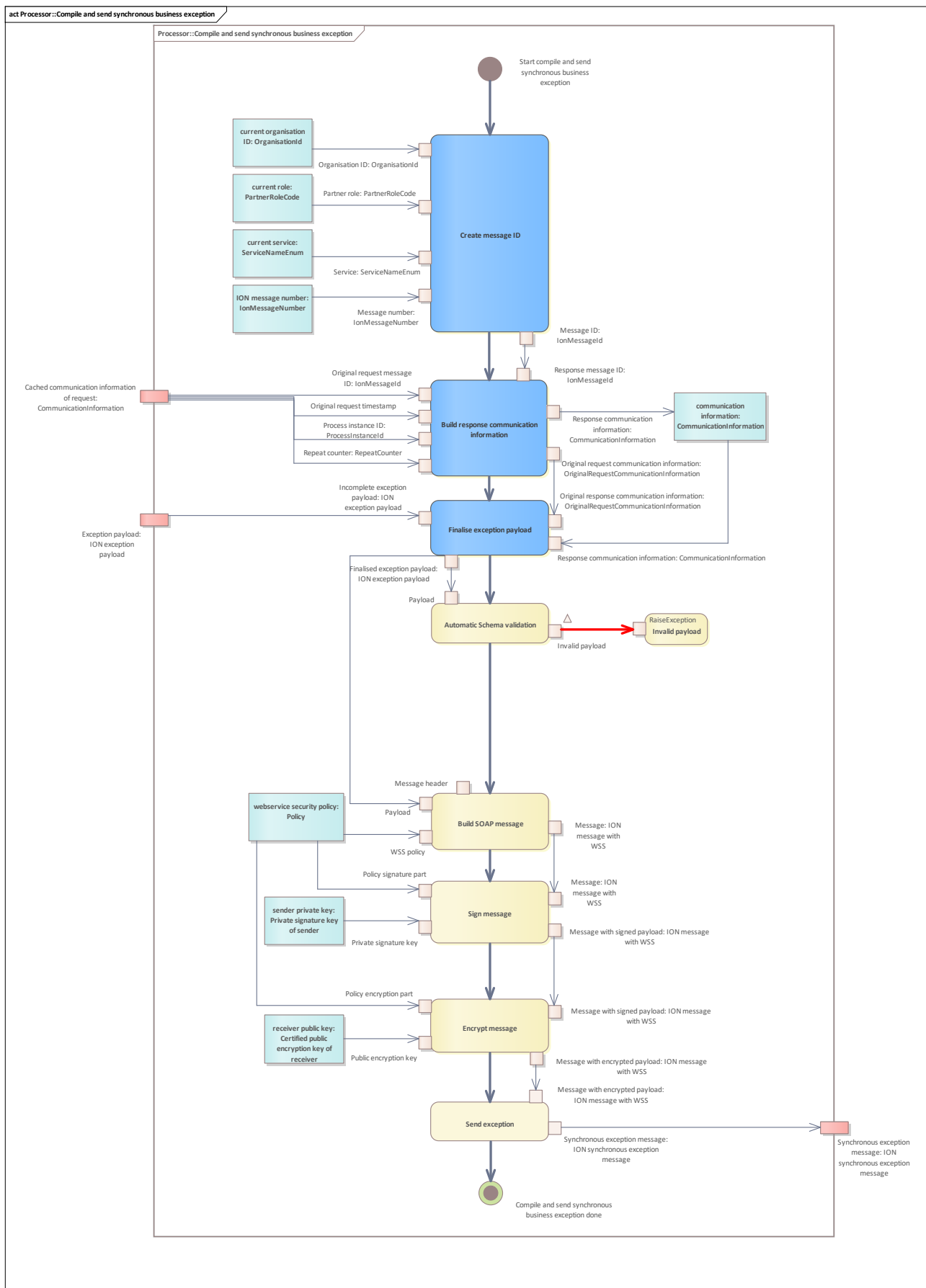
Base Activity	
Inputs	Exception payload : ION exception payload Cached communication information of request : CommunicationInformation
Outputs	Synchronous exception message : ION synchronous exception message
Error Cases	
Activity Diagram	Processor::Compile and send synchronous business exception

6.1.2.7.1 Processor::Compile and send synchronous business exception

Activity for [Compile and send synchronous business exception](#).



6.1.2.7.1.1 Processor::Compile and send synchronous business exception



Build SOAP message

Build the SOAP message employing a [SOAP Envelope](#) with consideration of the binding rules in the corresponding WSDL. These rules determine if an [IonRoutingHeader](#) has to be embedded in the [SOAP Header](#).

Furthermore, the embedded webservice security policy in the WSDL is used to sign and encrypt the necessary parts of the message.

Encrypt message

Encrypt the message with the receiver's public key. This public key is certified and can be obtained from the PKI by its directory service.

Note: please consider the rules concerning the [Validity of Certificates](#).

Due to the web service security [Policy](#), the entire [SOAP Body](#) has to be encrypted. The [SOAP Header](#) remains in clear text for routing purposes. Furthermore, the signature of the [SOAP Body](#) also has to be encrypted. The encrypted signature data of the signed [SOAP Body](#) is stored in the [Security header](#) in the [SOAP Header](#) and is not part of the [SOAP Body](#).

Sign message

Sign the message parts as defined in the [Webservice Security Policy](#) with the private key for signature purposes (the corresponding public key is registered and certified in the PKI and can be accessed via the directory service of the PKI).

According to the web service security specification, the entire body of the SOAP message which contains the payload is signed.

Normally performed by the underlying WSS framework.

Automatic Schema validation

Do the automatic XML schema validation with the embedded payload.

Normally performed by the underlying web service framework.

Create message ID

Create a new [IonMessageId](#) which is embedded in the communication information of the payload.

Build response communication information

Complete the "existing" payload's communication information.

To build the [CommunicationInformation](#) object of the response:

- sender ID from request [IonMessageId](#) -> response receiver ID
- sender role from request [IonMessageId](#) -> response receiver role
- sender service from request [IonMessageId](#) -> response receiver service
- new [IonMessageId](#) for the response
- new message timestamp
- no [RepeatCounter](#)!
- request [ProcessInstanceId](#) -> response process instance ID

Take the following fields from [CommunicationInformation](#) of the original request and copy them to the response payload to build the [OriginalRequestCommunicationInformation](#):

- [IonMessageId](#)
- Timestamp
- [RepeatCounter](#) (if present)

Finalise exception payload

Complete the "incomplete" [exception payload](#) by adding the [OriginalRequestCommunicationInformation](#) object and the newly built [CommunicationInformation](#) object.

Invalid payload

Exception if an automatic schema validation error occurs.

Two scenarios may occur:

- If the automatic schema validation of the incoming message fails, the software implementation of the sender has to be fixed. The called operation terminates with this exception and no processing takes place
- If the automatic schema validation of the outgoing message fails, your software implementation or data storage has to be fixed. The process stops with this exception and prevents the case from above

For incoming messages, the underlying framework has to respect the requirement [Security relevant errors](#) to avoid DoS attacks. If the framework does not fulfil it, a separate exception handler must be employed.

Send exception

Send the [synchronous ION exception message](#) with the signed and encrypted [exception payload](#).

6.1.2.8 Basic path with hidden CRE (explanatory purpose)

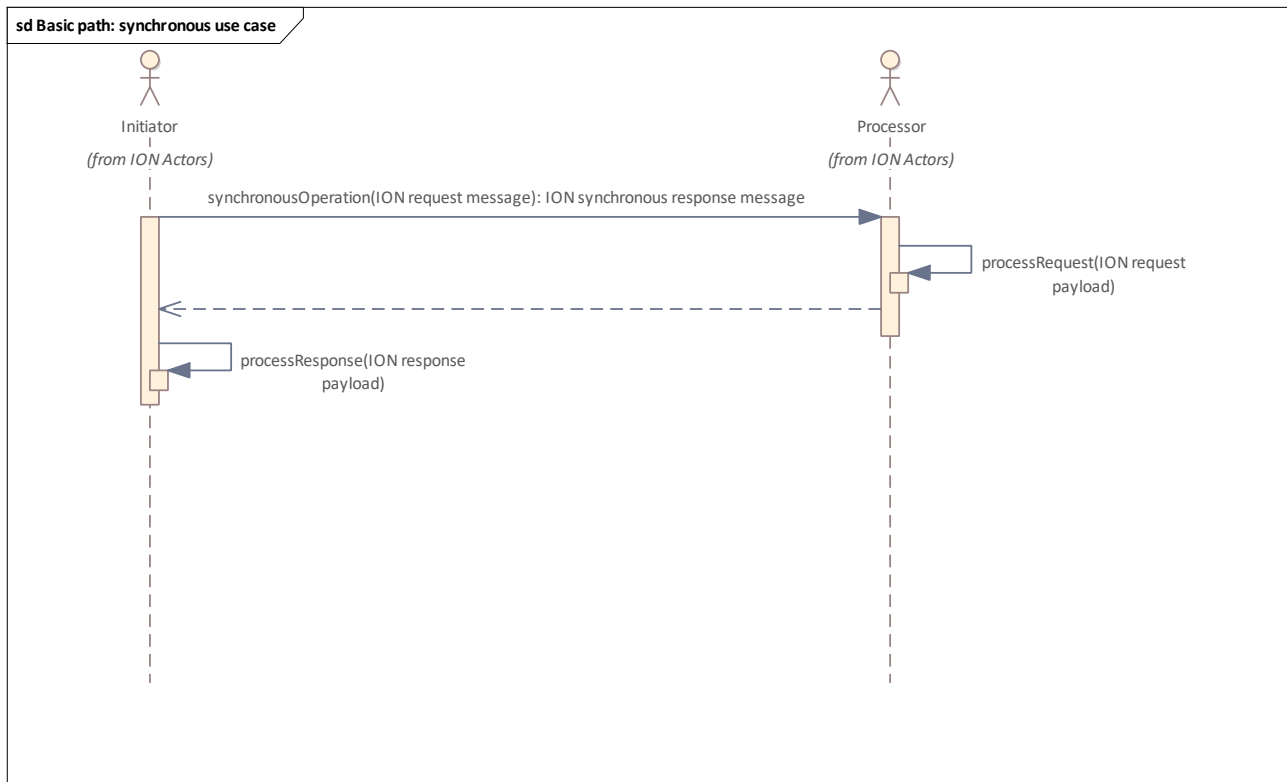
Sequence diagram which shows the basic flow of ION message exchange in a [Synchronous Use Case](#).

The diagram shows a synchronous message exchange scenario.

1. The [Initiator](#) starts with a new [ION request message](#).
2. The [Processor](#) takes this message, performs internal processing and sends a [Synchronous ION response message](#).
3. For notification scenarios (the initiator notifies the processor about an executed transaction), the [Response payload](#) contains a [BusinessAcknowledgement](#). For get/mixed scenarios, the [Response payload](#) may contain additional [Response business data](#).
4. The initiator processes the response and the use case ends.

This diagram hides the intermediary [Central routing engine](#) to show only the pure message exchange between [Initiator](#) and [Processor](#).

6.1.2.8.1 Basic path: synchronous use case



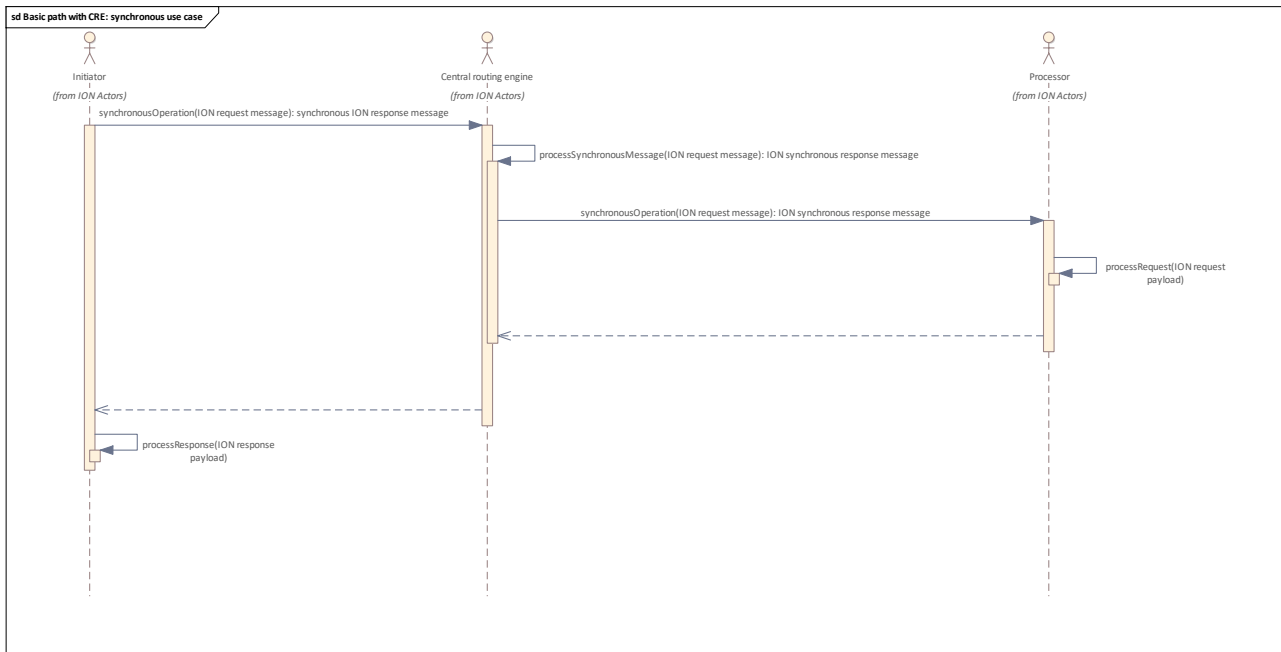
See [Basic path](#).

6.1.2.9 Basic path with CRE shown (real scenario)

Sequence diagram which shows the basic flow of ION message exchange in a [Synchronous Use Case](#).

In addition to the interaction in the [Basic path](#), this diagram also shows the intermediary [Central routing engine](#), to display the real message exchange between [Initiator](#) and [Processor](#).

6.1.2.9.1 Basic path with CRE: synchronous use case



See [Basic path with CRE](#).

6.1.2.10 Business exception path with hidden CRE (explanatory purpose)

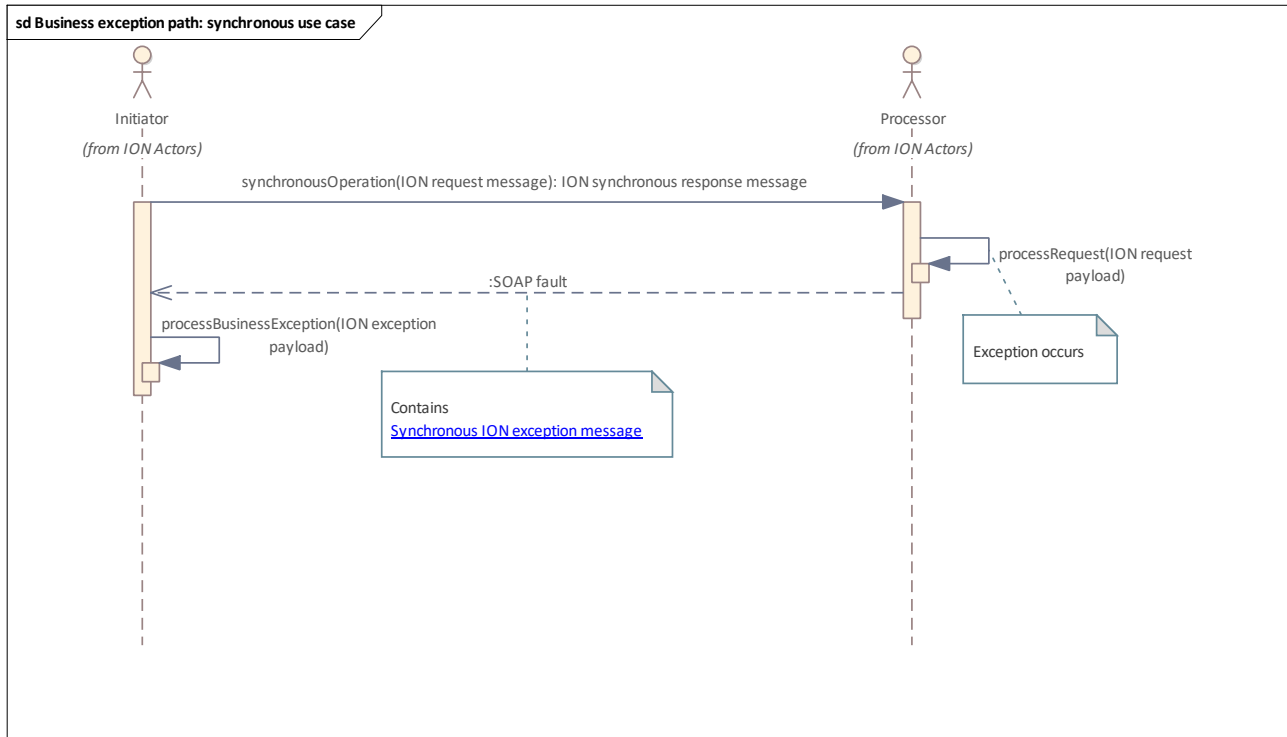
Sequence diagram which shows the business exception flow of ION message exchange in a [Synchronous Use Case](#).

The diagram shows a synchronous message exchange scenario.

1. The [Initiator](#) starts with a new [ION request message](#).
2. The [Processor](#) takes this message and performs its internal processing.
3. During the process, a check fails and a business exception is raised. Then the processor sends a [Synchronous ION exception message](#).
4. The initiator processes the exception and the use case ends.

This diagram hides the intermediary [Central routing engine](#) to show only the pure message exchange between [Initiator](#) and [Processor](#).

6.1.2.10.1 Business exception path: synchronous use case



See [Business exception path](#).

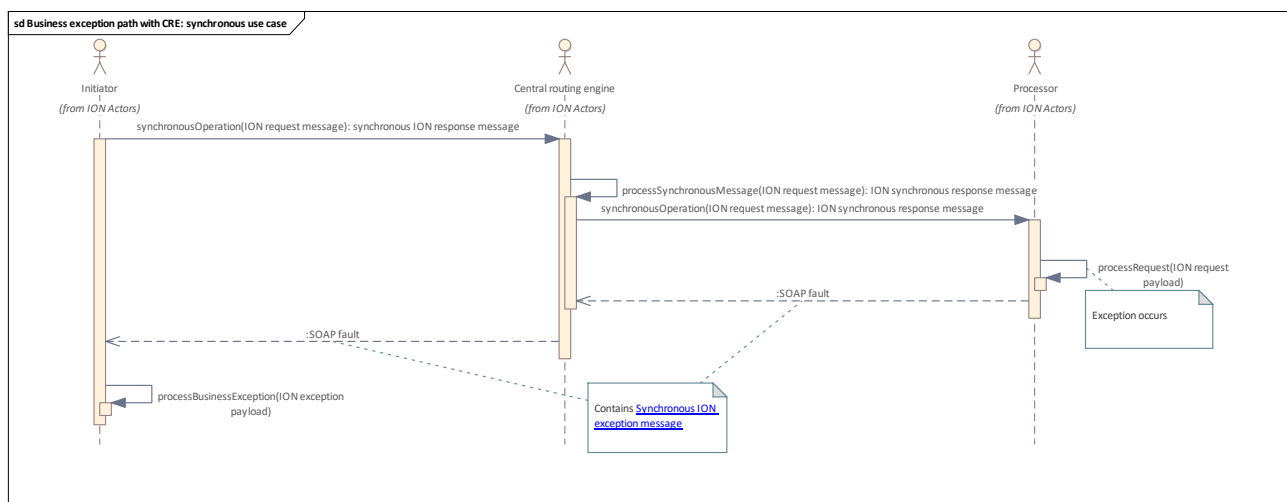
6.1.2.11 Business exception path with CRE shown (real scenario)

Sequence diagram which shows the business exception flow of ION message exchange in a [Synchronous Use Case](#).

The diagram shows a synchronous message exchange scenario.

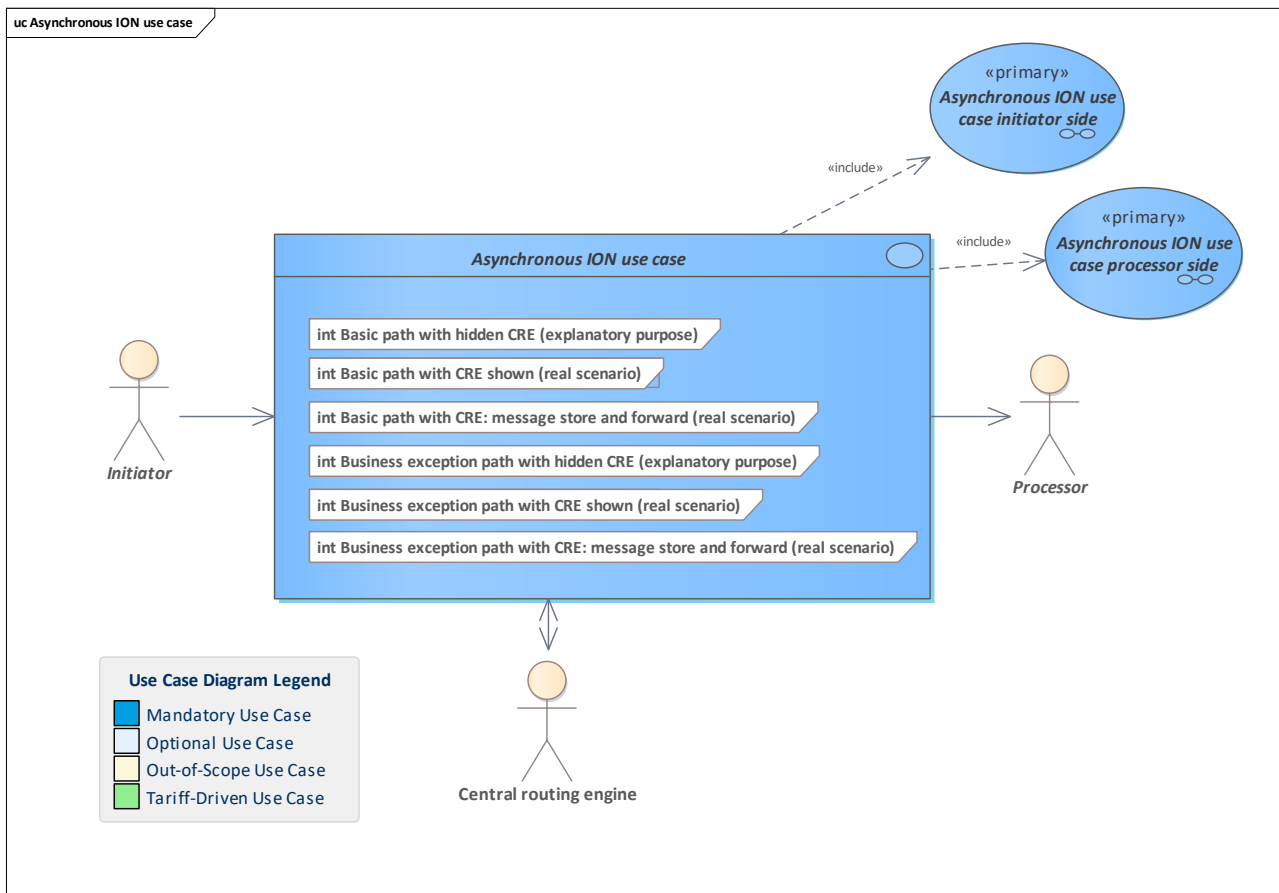
In addition to the interaction [Business exception path](#), this diagram also shows the intermediary [Central routing engine](#), to display the real message exchange between [Initiator](#) and [Processor](#).

6.1.2.11.1 Business exception path with CRE: synchronous use case



See [Business exception path with CRE](#).

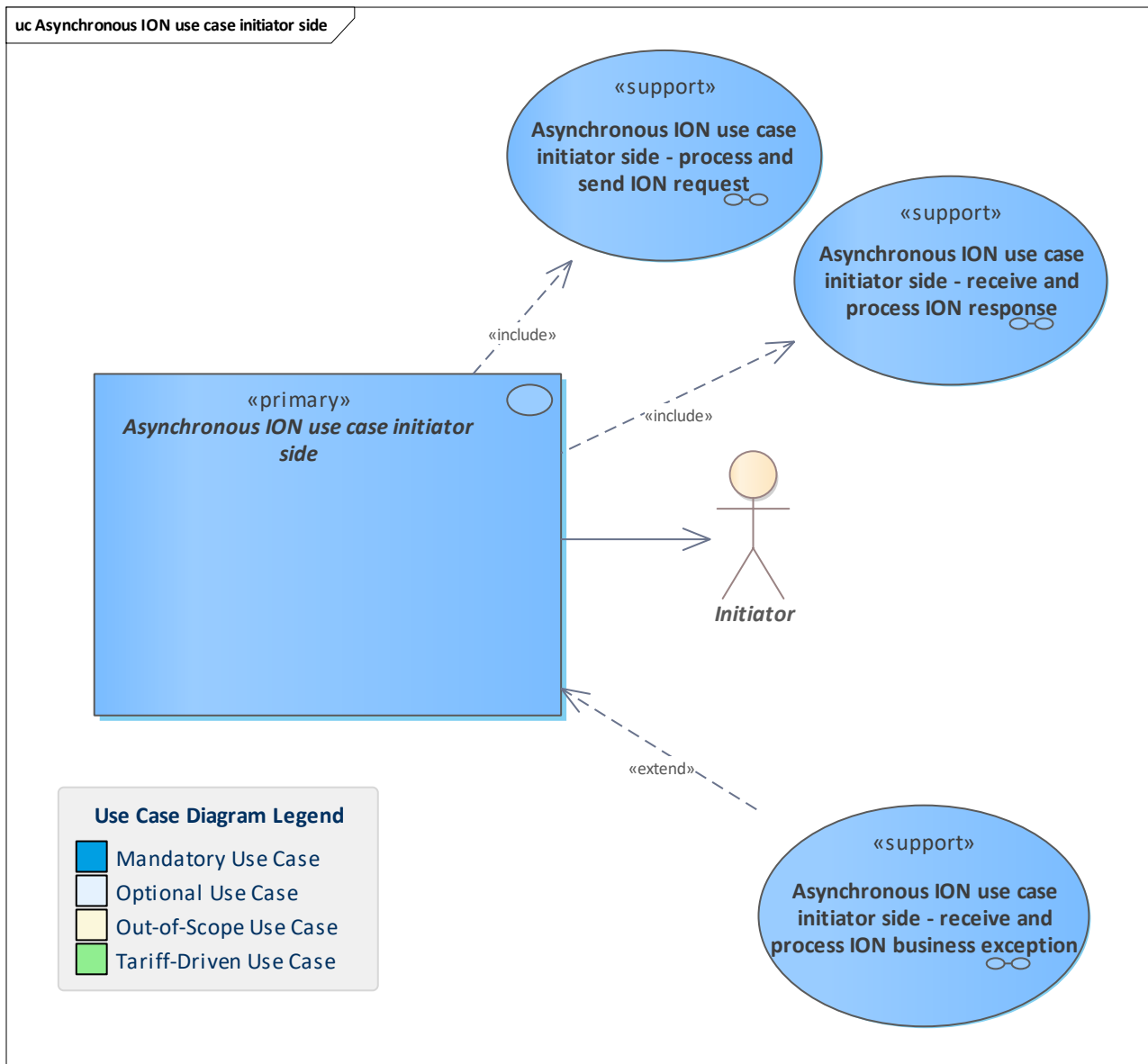
6.1.3 Asynchronous ION use case



Use Case Description	Asynchronous ION use case
	<p>General use case which meets the requirements for a workflow distributed over two systems involving asynchronous interactions. The asynchronous use case splits into the side of the initiator and the side of the processor.</p> <p>Each asynchronous use case works in the same way with a fixed workflow.</p> <p>The asynchronous behaviour is achieved by employing two synchronous calls with a DeliveryAcknowledgement, one for the request sent by Initiator to the Processor and one for the later response sent by the processor to the initiator.</p> <ol style="list-style-type: none"> 1. In the first iteration, the Initiator sends a message to the Processor and gets a temporary (use case not yet finished) DeliveryAcknowledgement. 2. Use case with BusinessAcknowledgement as response: the initiator expects a business acknowledgement only, which is sent asynchronously by the processor. The initiator itself acknowledges this response with a delivery acknowledgement and then terminates the whole use case between the initiator and processor. 3. Use case with further data in the response: The initiator expects a qualified business response in the form of a response payload, which contains further Response business data in addition to the business acknowledgement. As in the scenario

	<p>above, the qualified response is sent later (asynchronously) by the processor and then acknowledged with a delivery acknowledgement by the initiator.</p> <p>Important Note: The technical roles of client and server are switched during the flow described above, however, the Initiator always remains the initiator and the Processor always remains the processor. This behaviour is emphasized by the arrows in the associations between the actors and the use case.</p>
Initiating Actor	Initiator Central routing engine
Reacting Actor	Processor Central routing engine
Preconditions	
Postconditions	
Linked Use Cases (Extended By)	
Linked Use Cases (Includes)	Asynchronous ION use case processor side / Asynchronous ION use case initiator side
Linked Use Cases (Realises)	
Base Activity	
Inputs	
Outputs	
Error Cases	
Activity Diagram	

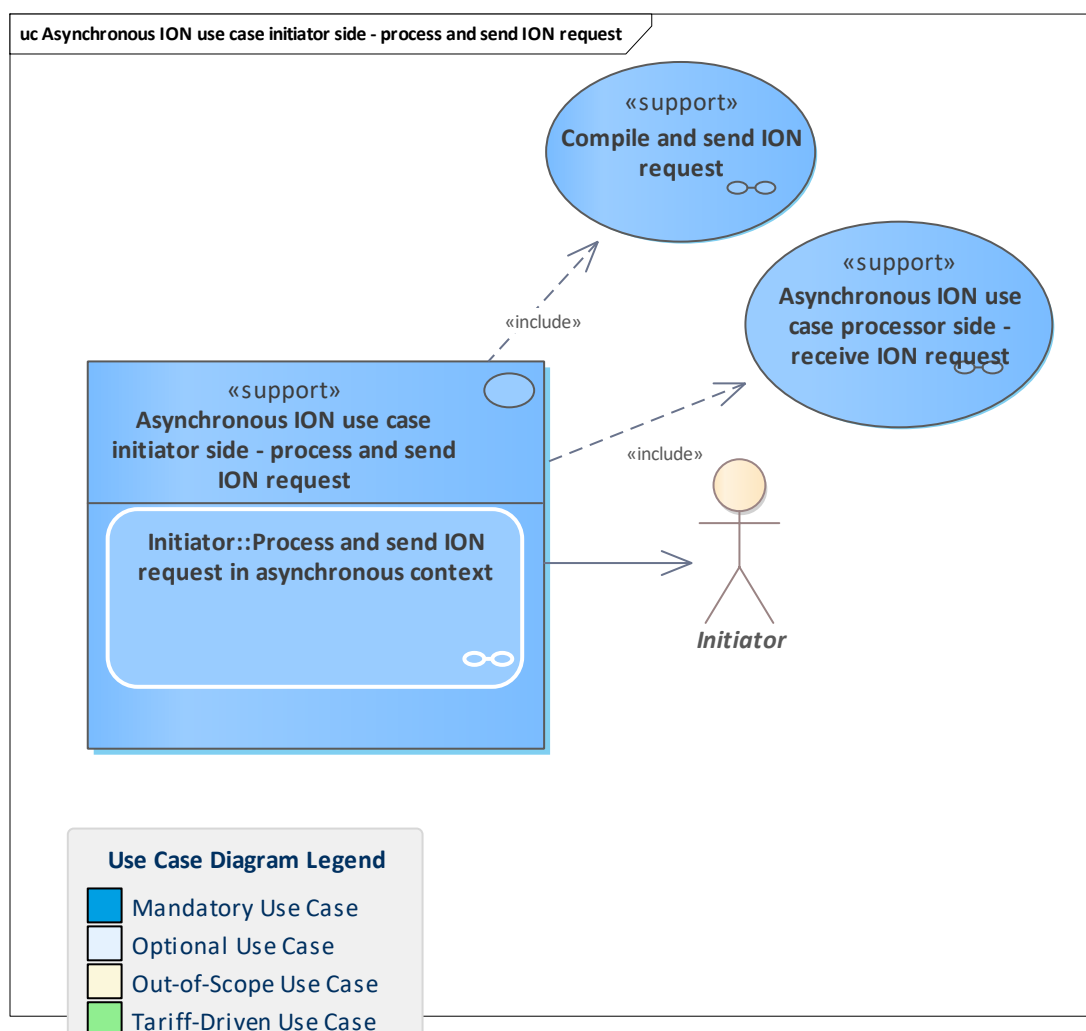
6.1.3.1 Asynchronous ION use case initiator side



Use Case	Asynchronous ION use case initiator side
Description	<p>This use case covers the whole initiator side in an asynchronous use case started by the Initiator.</p> <p>The steps are:</p> <ul style="list-style-type: none"> Perform the business logic and collect business data for a request Build the suitable ION request and send it to the Processor. Get the temporary DeliveryAcknowledgement and wait for the final response Receive the asynchronous ION response and perform all common checks Send a DeliveryAcknowledgement to the processor. Go through the remaining business logic to process the response depending on its content
Initiating Actor	
Reacting Actor	Initiator
Preconditions	
Postconditions	

Linked Use Cases (Extended By)	Asynchronous ION use case initiator side - receive and process ION business exception
Linked Use Cases (Includes)	Asynchronous ION use case initiator side - receive and process ION response / Asynchronous ION use case initiator side - process and send ION request
Linked Use Cases (Realises)	
Base Activity	
Inputs	
Outputs	
Error Cases	
Activity Diagram	

6.1.3.2 Asynchronous ION use case initiator side - process and send ION request



Use Case	Asynchronous ION use case initiator side - process and send ION request
Description	This use case executes the first part of an Asynchronous ION use

	case initiator side . An asynchronous ION use case consists of two synchronous operation calls. Part 1: do any business logic, compile a request, send it to the Processor and store the deliveryAcknowledgement . Consider this part 1 for the SLA timeout. Within a certain time span, a reply is expected.
Initiating Actor	
Reacting Actor	Initiator
Preconditions	
Postconditions	
Linked Use Cases (Extended By)	
Linked Use Cases (Includes)	Asynchronous ION use case processor side - receive ION request / Compile and send ION request
Linked Use Cases (Realises)	
Base Activity	
Inputs	
Outputs	
Error Cases	
Activity Diagram	Initiator::Process and send ION request in asynchronous context

6.1.3.2.1 Initiator::Process and send ION request in asynchronous context

Activity that shows sample steps and checks on the initiator-side in an asynchronous context. This activity ends when the initiator-side business logic has finished and the message has to be sent from the [Initiator](#) to the [Processor](#).

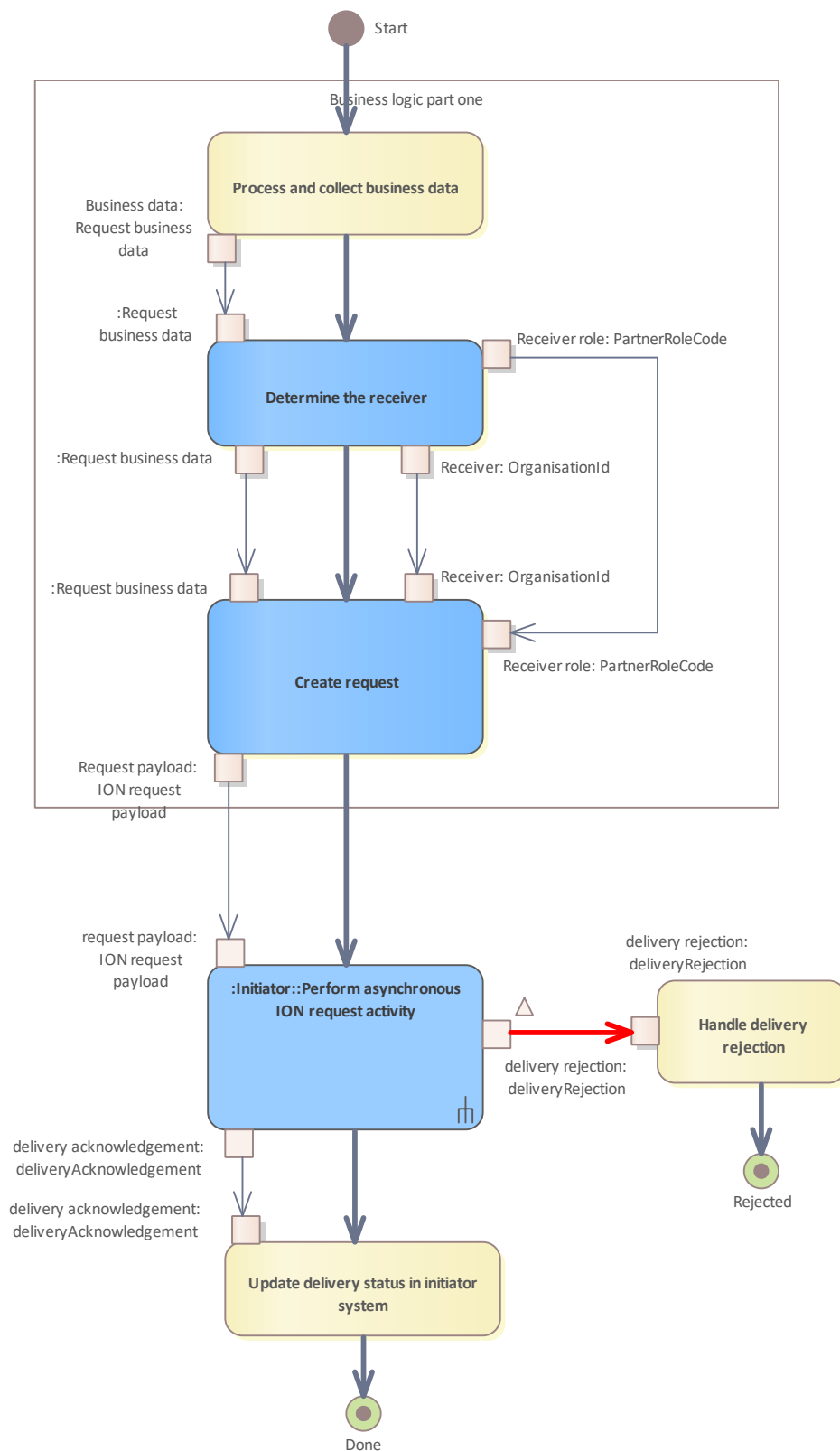


**6.1.3.2.1.1
context**

Initiator::Process and send ION request in asynchronous

act Initiator::Process and send ION request in asynchronous context

Initiator::Process and send ION request in asynchronous context



Activity diagram which shows the control flow, object flow and potential exception flow triggered by an [Asynchronous use case client side](#).

Process and collect business data

Abstract action that stands for the concrete business logic which collects the necessary data in one or more steps.

If no business data is needed (e.g. for pure get-scenarios), this step will only indicate which data is needed from a third-party system (getXXX).

Determine the receiver

E.g. in notification scenarios, the receiver organisation can be extracted from the contained binary transaction attestation. The receiver role comes from the use case context (e.g. a notification about an entitlement blocking is always initially sent to the PO) based on the information of the intended called operation that was called.

Note: One organisation may have more than one role.

Create request

Create the request as an XSD top level element containing the [Request business data](#), together with the [CommunicationInformation](#) as far as possible. Do not consider the creation of the [IonMessageId](#) or [ProcessInstanceId](#). This is done later during the ION message compilation. The own organisation ID, as well as the own role, is also added later in the ION message compilation, see [Compile and send ION request](#).

The receiver role (and service) has to be set only if the use case does not determine it.

Update delivery status in initiator system

Register [deliveryAcknowledgement](#) as preliminary confirmation, so that the request can be processed in the [Processor](#)'s business logic. The next step is to wait the SLA timespan for the final business response (see [\[5\] SLAs for Message Transfer](#)).

Handle delivery rejection

Action that handles a potential [deliveryRejection](#). This reply has to be evaluated and then, the right actions must be taken depending on the use case.

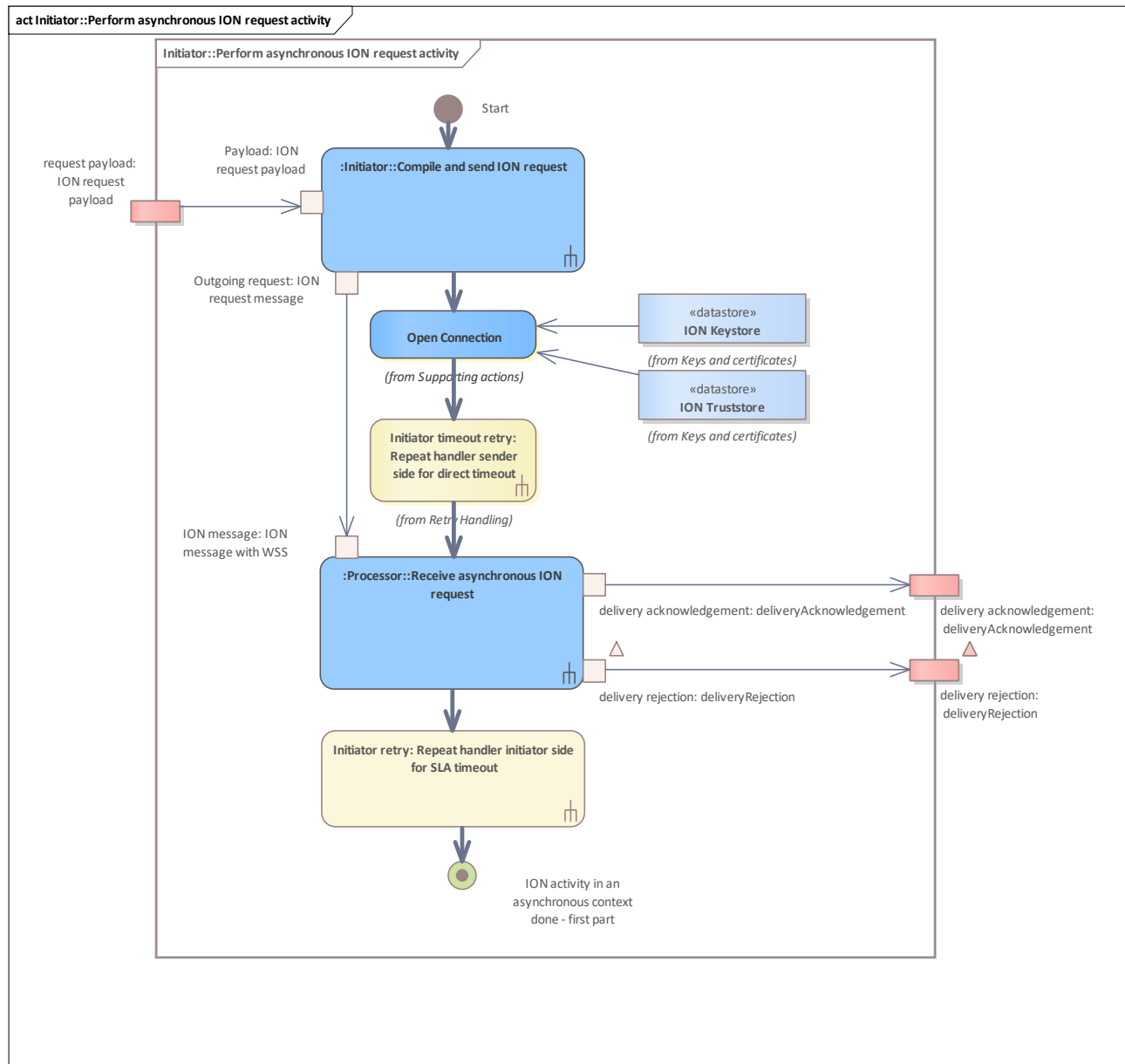
Initiator::Perform asynchronous ION request activity

See [Initiator::Perform asynchronous ION request activity](#).

6.1.3.2.2 Initiator::Perform asynchronous ION request activity

Activity that shows the common steps and checks to be done if the ION communication on the initiator side is involved in an asynchronous context. This activity starts when the initiator-side business logic has finished and the message has to be sent from the [Initiator](#) to the [Processor](#).

6.1.3.2.2.1 Initiator::Perform asynchronous ION request activity



See [Initiator::Perform ION request activity in an asynchronous context](#).

Initiator::Compile and send ION request

See [Initiator::Compile and send ION request](#).

Processor::Receive asynchronous ION request

See [Processor::Receive asynchronous ION request](#).

Repeat handler initiator side for SLA timeout

See [Repeat handler initiator side for SLA timeout](#).

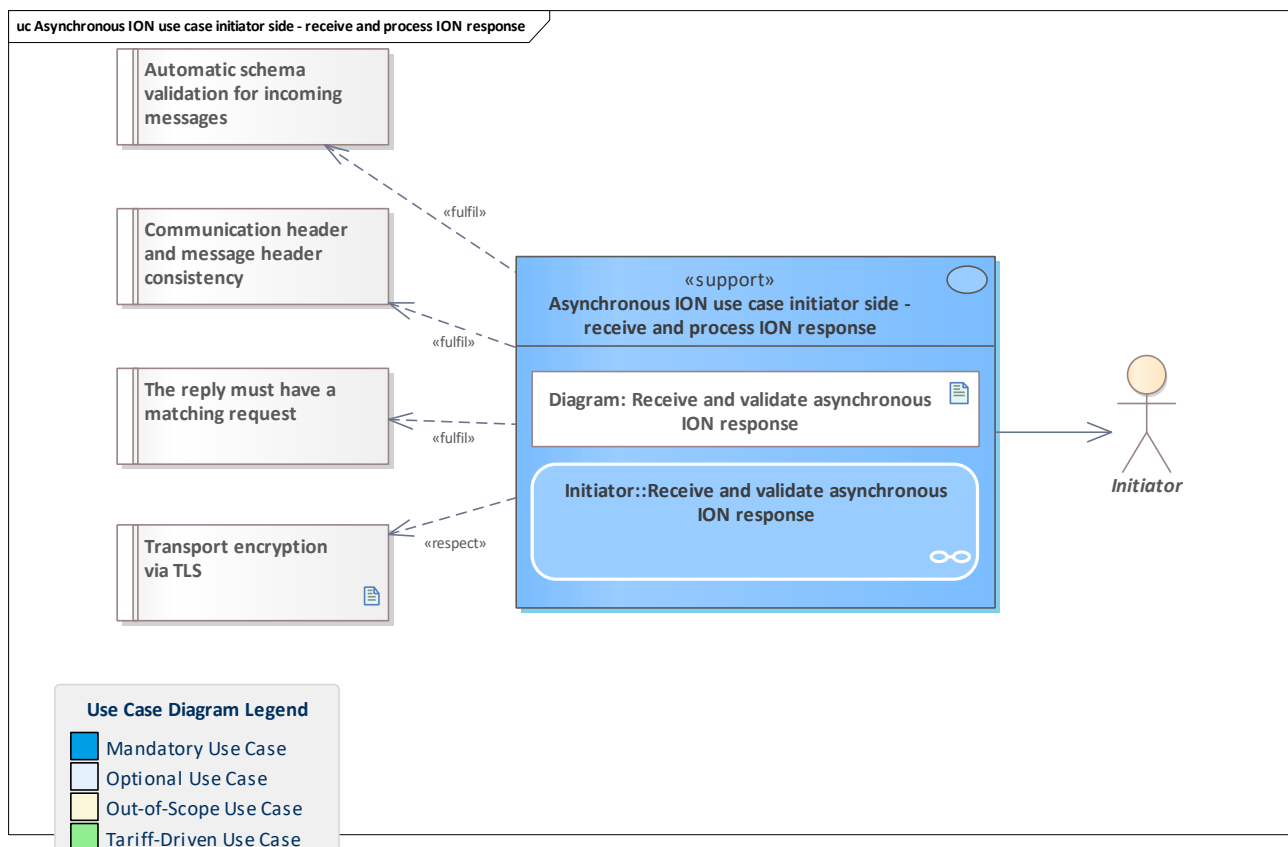
Repeat handler sender side for direct timeout

See [Repeat handler sender side for direct timeout](#).

Open Connection

Opens a TLS connection (to the CRE) presenting the [ION TLS certificate](#) . In case the CRE URL uses the HTTP scheme, open TCP connection or re-use an existing one.
Opens a TLS connection to the CRE presenting the [ION TLS certificate](#) found in the [ION Keystore](#) (see also [Keys and certificates : ION Key- and Truststore](#)) for the configured organisation ID. Checks the validity of the retrieved certificate against the [ION Truststore](#) (incorporating CRLs or OCSP). Asserts that the organisation ID given in the "o" part of the certificate subject matches 5602 (organisation ID of the CRE) and that there is no "ou" part (i.e. it is an TLS certificate).

6.1.3.3 Asynchronous ION use case initiator side - receive and process ION response



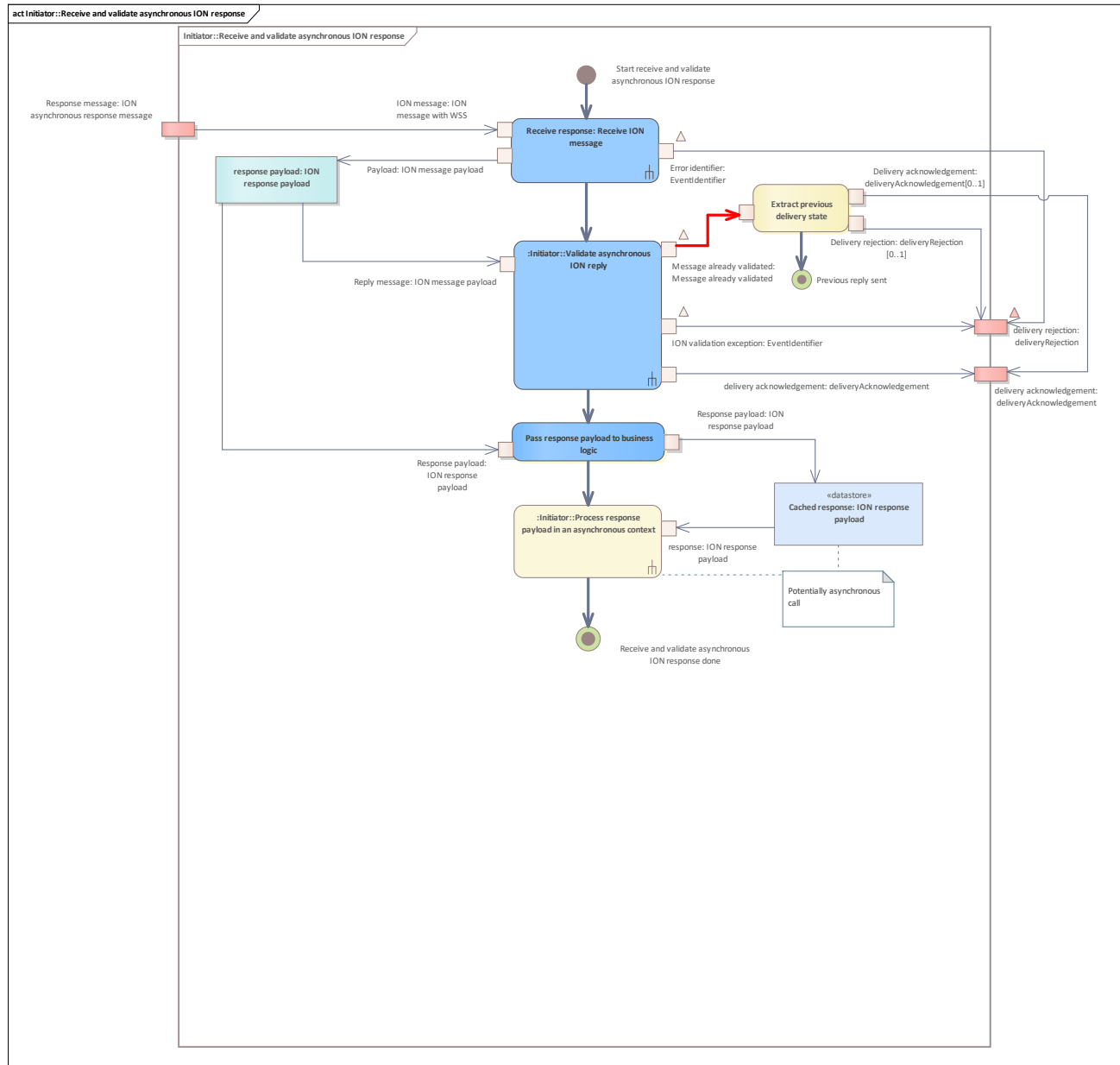
Use Case	Asynchronous ION use case initiator side - receive and process ION response
Description	<p>A common use case for receiving a message. To emphasize the business direction (Processor to Initiator), the message is called response.</p> <p>This use case is only used in the asynchronous context.</p> <p>Part 2a (for part 1, see Asynchronous ION use case initiator side - process and send ION request): receive either a BusinessAcknowledgement only or combined with further business data coming from the processor. The response is wrapped with a suitable use case-specific top-level element wrapper. This wrapper</p>

	<p>is named after the operation provided by the Processor which was called by the Initiator, extended by the keyword "response". For example, notifyXY will be notifyXYResponse and this operation is provided by the Initiator.</p> <p>The Initiator has to ensure that the message is authentic and that the syntax and content of the response are valid.</p> <p>This scenario is valid for all derived use cases which have to receive asynchronous responses.</p> <p>For the BusinessAcknowledgement part of the response, the main step is to find the reference of the original transaction. This reference is delivered in the BusinessAcknowledgement.</p> <p>If the reference cannot be found, an E IONC ORIGINAL REQUEST NOT FOUND exception is generated as a deliveryRejection, since the parent use case ends with this acknowledgement.</p> <p>The same applies in the case of a previously delivered identical response. In this case, E ION DUPLICATE RESPONSE is used.</p> <p>If possible additional business data should cause problems in the initiator's system, no external exception is generated, as the parent use case ends with the above acknowledgement and the additional business data could possibly only be processed asynchronously after a deliveryRejection has already been sent. This problem has then to be treated internally or with the notifyEvents method of the processor.</p>
Initiating Actor	
Reacting Actor	Initiator
Preconditions	
Postconditions	
Linked Use Cases (Extended By)	
Linked Use Cases (Includes)	
Linked Use Cases (Realises)	
Base Activity	
Inputs	Response message : ION asynchronous response message
Outputs	delivery acknowledgement : deliveryAcknowledgement
Error Cases	E ION DATA OF SOAP HEADER DOES NOT MATCH MESSAGE DATA E ION OPERATION NOT IMPLEMENTED E ION RECEIVER ORGANISATION MISMATCH E ION RECEIVER ROLE MISMATCH E ION RECEIVER SERVICE MISMATCH E ION UNKNOWN SENDER E ION WRONG SENDER ROLE E ION ORIGINAL REQUEST NOT FOUND E ION DUPLICATE RESPONSE E ION DUPLICATE ION MESSAGE ID E ION DUPLICATE ION MESSAGE ID delivery rejection : deliveryRejection
Activity Diagram	Initiator::Receive and validate asynchronous ION response

6.1.3.3.1 Initiator::Receive and validate asynchronous ION response

See [Receive and validate asynchronous ION response](#).

6.1.3.3.1.1 Initiator::Receive and validate asynchronous ION response



Activity diagram which shows the control flow, object flow and potential exception flow triggered by the use case [Receive and validate asynchronous ION response](#).

Pass response payload to business logic

Allow the response - represented by the response payload - to be returned to the higher-level business logic after the generic checks have been performed.

Receive ION message

See [Receive ION message](#).

Initiator::Process response payload in an asynchronous context

See [Initiator::Process response payload in an asynchronous context](#).

Extract previous delivery state

Extract the previous reply concerning the delivery state of the message. Return either the contained [deliveryAcknowledgement](#) or [deliveryRejection](#).

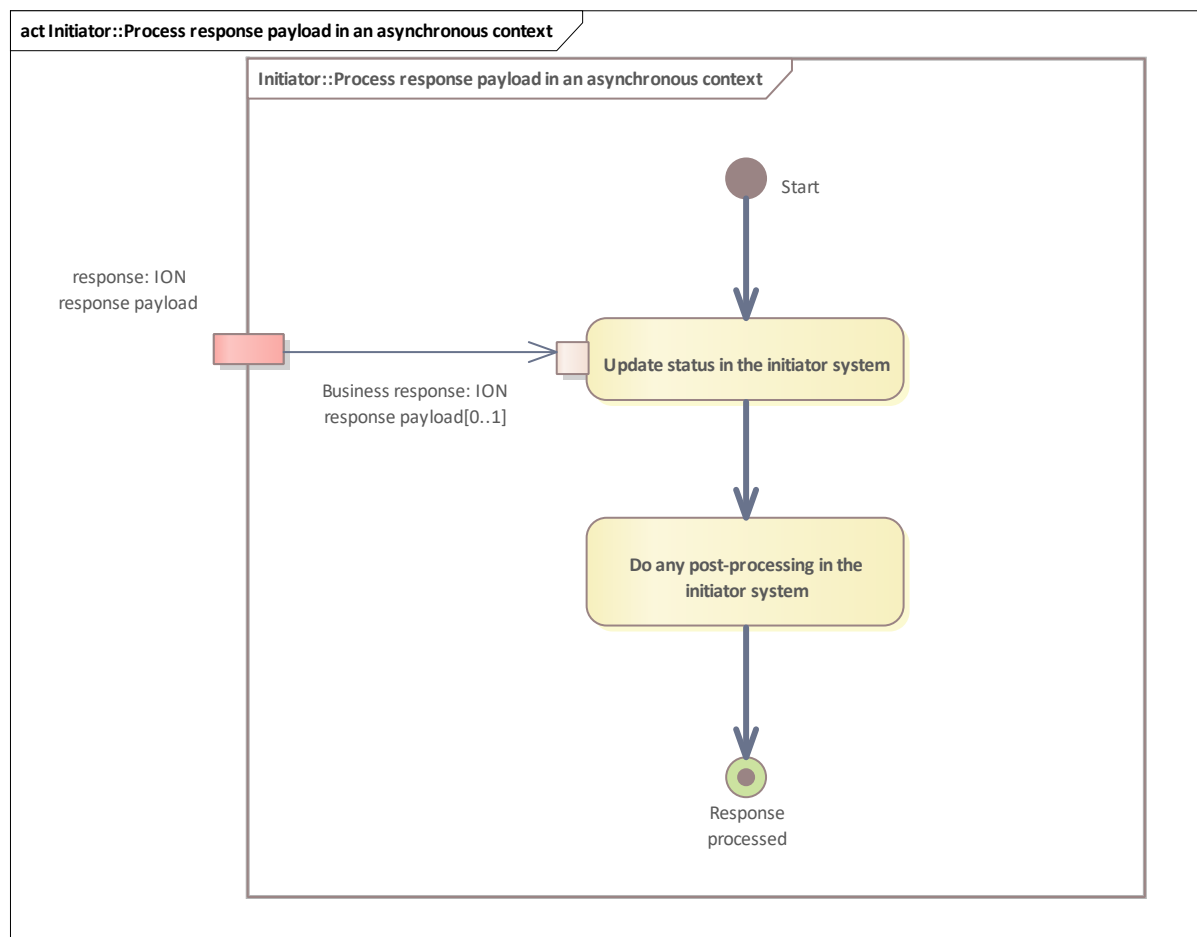
Initiator::Validate asynchronous ION reply

See [Initiator::Validate asynchronous ION reply](#).

6.1.3.3.2 Initiator::Process response payload in an asynchronous context

Activity that includes the processing of any [Response business data](#) in an asynchronous context.

6.1.3.3.2.1 Initiator::Process response payload in an asynchronous context



Activity diagram for [Initiator::Process response payload in asynchronous context](#).

Update status in the initiator system

A business response or exception is received and status is updated accordingly in the initiator's system.

The generic message correlation steps are done before in the use cases [Receive and validate asynchronous ION response](#), [Receive and validate asynchronous business exception](#), [Receive and validate synchronous ION response](#) or [Receive and validate synchronous ION business exception](#).

This step should update the status of a certain entity depending on the underlying use case (e.g. set the state of an entitlement to "hotlisted").

In the case of a business exception: depending on exception type, a subsequent handling may be required after updating the status to enable regular processing in further actions. **It is assumed that the problem defined in the exception is solved before the action completes. Only then, the follow-up steps can continue.**

In the case of warnings contained in the reply, a further warning handling could be necessary. If the [response payload](#) contains [Response business data](#), this business data must be processed in further steps.

Do any post-processing in the initiator system

Do post-processing depending on the underlying use case.

If the [Response payload](#) contains [Response business data](#), this business data must be processed here.

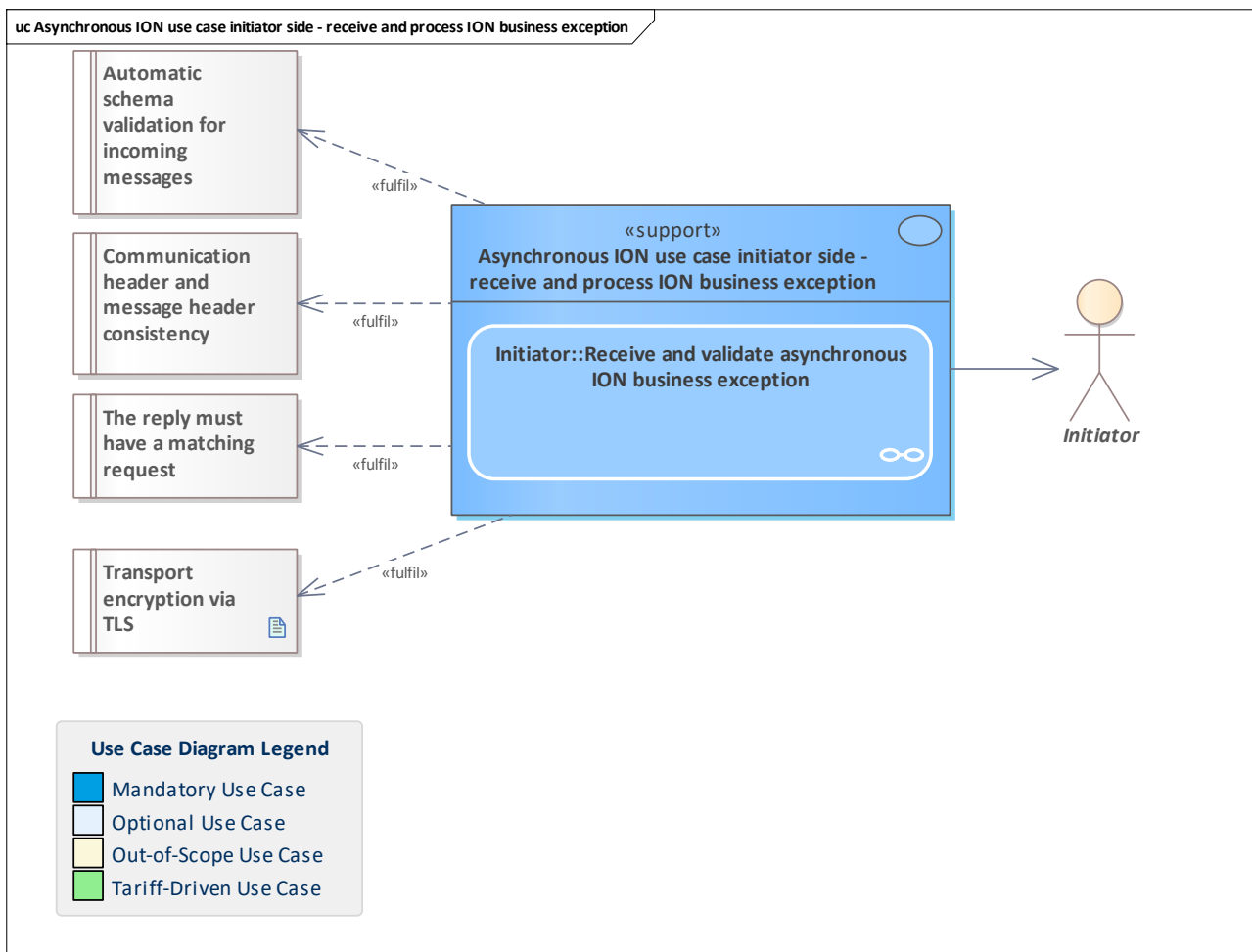
Any other post-processing is also possible here. This might be something out of scope like sub-ledger steps or something specified in etiCORE like further ION calls or calling an included use case.

6.1.3.3.3 Diagram: Receive and validate asynchronous ION response

A diagram that shows the supporting use case performed by the [Initiator](#) to receive and validate an asynchronous ION response. This use case contains all common steps and checks to be taken before the message is passed to the [Initiator](#) system's business logic.

Furthermore, the diagram shows the requirements to be fulfilled by the use case.

6.1.3.4 Asynchronous ION use case initiator side - receive and process ION business exception



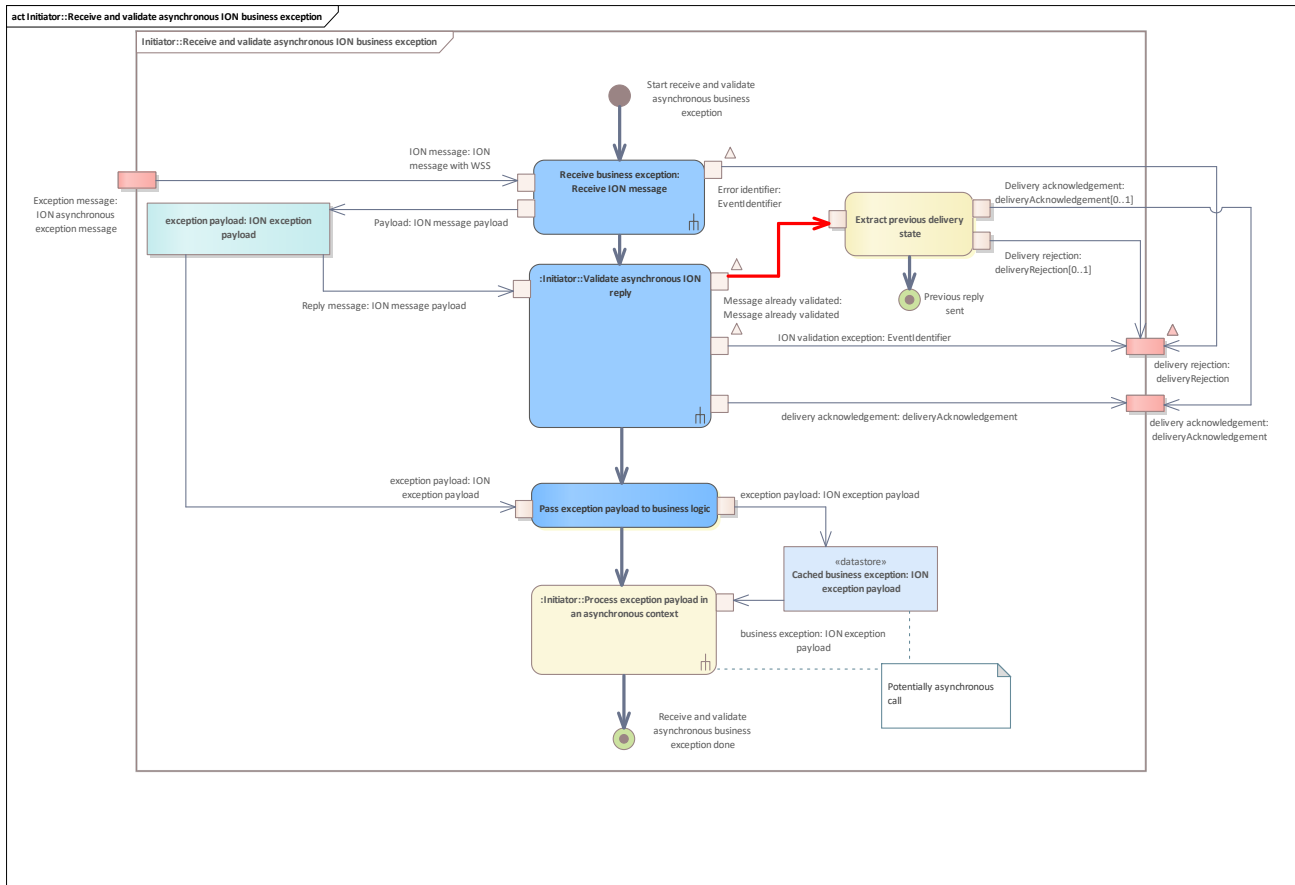
Use Case	Asynchronous ION use case initiator side - receive and process ION business exception
Description	<p>Common use case to receive an asynchronous ION exception message containing a BusinessException. This use case is only used in the asynchronous context. This scenario is valid for all derived use cases which have to receive asynchronous business exceptions.</p> <p>Part 2b (for part 1, see Asynchronous ION use case initiator side - process and send ION request):</p> <p>Receive an asynchronous ION exception message wrapped with a suitable use case-specific top-level element wrapper. This wrapper is named after the operation provided by the Processor which was called by the Initiator, extended by the keyword "Exception". For example, notifyXY will be notifyXYException and this operation is provided by the Initiator.</p> <p>The exception can be considered as a response in the form of a BusinessException which has an additional Error to be evaluated and possibly handled afterwards.</p> <p>Since no further business data is contained, the main step is to find the reference of the original transaction. This reference is delivered in the business exception.</p> <p>If the reference cannot be found, an E IONC ORIGINAL REQUEST NOT FOUND exception is generated as a deliveryRejection, since the parent use case ends with this</p>

	acknowledgement. The same applies in the case of a previously delivered identical response. In this case, E ION DUPLICATE RESPONSE is used.
Initiating Actor	
Reacting Actor	Initiator
Preconditions	
Postconditions	
Linked Use Cases (Extended By)	
Linked Use Cases (Includes)	
Linked Use Cases (Realises)	
Base Activity	
Inputs	Exception message : ION asynchronous exception message
Outputs	delivery acknowledgement : deliveryAcknowledgement
Error Cases	E ION DATA OF SOAP HEADER DOES NOT MATCH MESSAGE DATA E ION OPERATION NOT IMPLEMENTED E ION RECEIVER ORGANISATION MISMATCH E ION RECEIVER ROLE MISMATCH E ION RECEIVER SERVICE MISMATCH E ION UNKNOWN SENDER E ION WRONG SENDER ROLE E ION ORIGINAL REQUEST NOT FOUND E ION DUPLICATE RESPONSE E ION DUPLICATE ION MESSAGE ID E ION DUPLICATE ION MESSAGE ID delivery rejection : deliveryRejection
Activity Diagram	Initiator::Receive and validate asynchronous ION business exception

6.1.3.4.1 Initiator::Receive and validate asynchronous ION business exception

See [Receive and validate asynchronous business exception](#).

6.1.3.4.1.1 Initiator::Receive and validate asynchronous ION business exception



Activity diagram which shows the control flow, object flow and potential exception flow triggered by the use case [Receive and validate asynchronous business exception](#).

Pass exception payload to business logic

Allow the exception - represented by the exception payload payload - to be returned to the higher- level business logic after the generic checks failed.

Receive ION message

See [Receive ION message](#).

Initiator::Process exception payload in an asynchronous context

See [Initiator::Process exception payload in an asynchronous context](#).

Extract previous delivery state

Extract the previous reply concerning the delivery state of the message. Return either the contained [deliveryAcknowledgement](#) or [deliveryRejection](#).

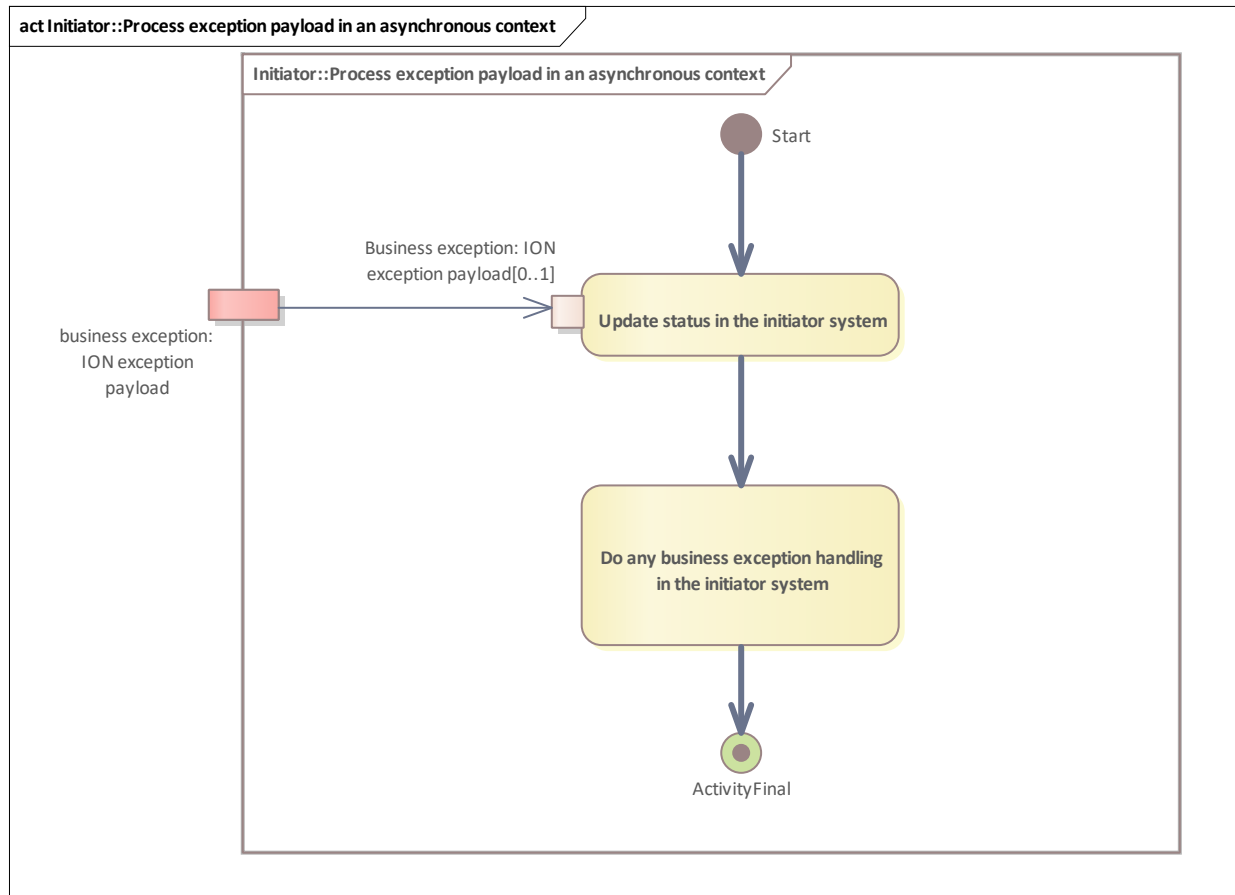
Initiator::Validate asynchronous ION reply

See [Initiator::Validate asynchronous ION reply](#).

6.1.3.4.2 Initiator::Process exception payload in an asynchronous context

Activity that includes the processing of any [exception payload](#) in an asynchronous context.

6.1.3.4.2.1 Initiator::Process exception payload in an asynchronous context



Activity diagram for [Initiator::Process exception payload in an asynchronous context](#).

Update status in the initiator system

A business response or exception is received and status is updated accordingly in the initiator's system.

The generic message correlation steps are done before in the use cases [Receive and validate asynchronous ION response](#), [Receive and validate asynchronous business exception](#), [Receive and validate synchronous ION response](#) or [Receive and validate synchronous ION business exception](#).

This step should update the status of a certain entity depending on the underlying use case (e.g. set the state of an entitlement to "hotlisted").

In the case of a business exception: depending on exception type, a subsequent handling may be required after updating the status to enable regular processing in further actions. **It is assumed that the problem defined in the exception is solved before the action completes. Only then, the follow-up steps can continue.**

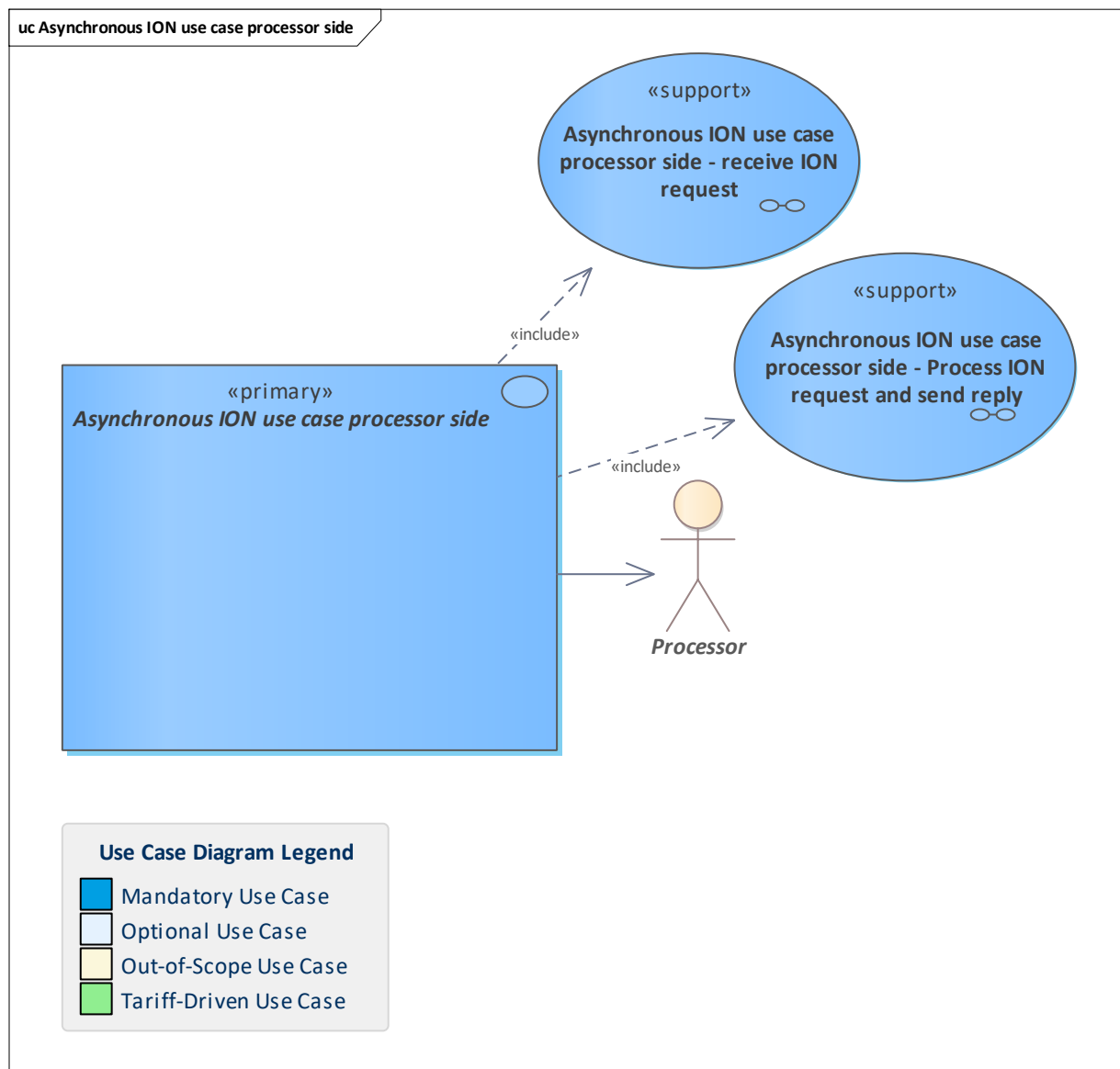
In the case of warnings contained in the reply, a further warning handling could be necessary.

If the [response payload](#) contains [Response business data](#), this business data must be processed in further steps.

Do any business exception handling in the initiator system

After an [asynchronous ION exception message](#) was received and acknowledged, there may be - depending on the kind of exception - necessary post processing, potentially asynchronously, too.

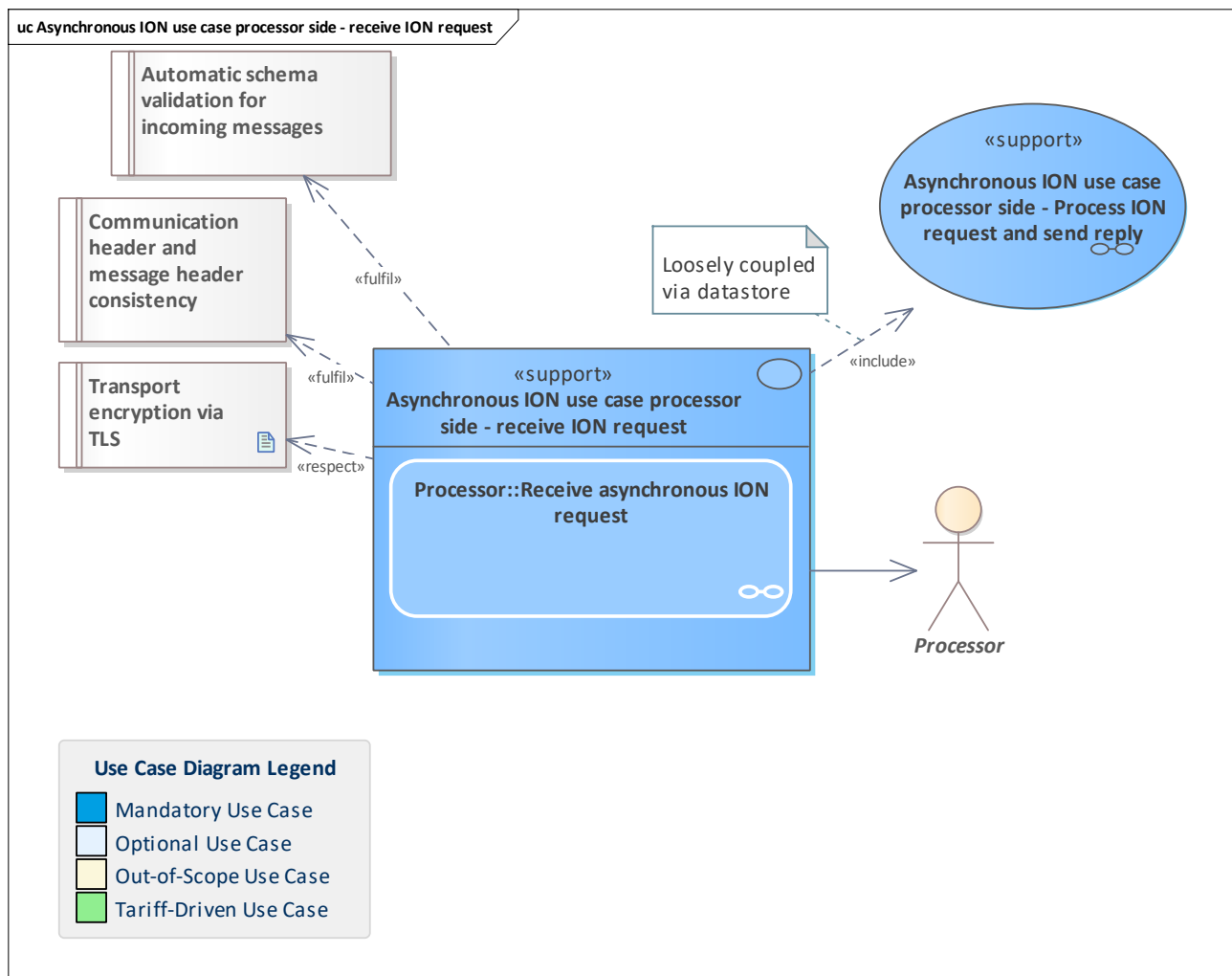
6.1.3.5 Asynchronous ION use case processor side



Use Case	Asynchronous ION use case processor side
Description	<p>This use case covers the whole processor side in an asynchronous use case context performed by the Processor.</p> <p>The steps are:</p> <ul style="list-style-type: none"> Receive and validate ION request Send a temporary DeliveryAcknowledgement to the Initiator Perform the business logic and, depending on the use case,

	collect business data for a response <ul style="list-style-type: none"> • Compile and send asynchronous ION response • Receive the final DeliveryAcknowledgement and update the status
Initiating Actor	
Reacting Actor	Processor
Preconditions	
Postconditions	
Linked Use Cases (Extended By)	
Linked Use Cases (Includes)	Asynchronous ION use case processor side - Process ION request and send reply / Asynchronous ION use case processor side - receive ION request / Asynchronous ION use case processor side - Process ION request and send reply
Linked Use Cases (Realises)	
Base Activity	
Inputs	
Outputs	
Error Cases	
Activity Diagram	

6.1.3.6 Asynchronous ION use case processor side - receive ION request



Use Case	Asynchronous ION use case processor side - receive ION request
Description	A common use case for receiving a message. To emphasize the business direction (Initiator to Processor), the message is called "request". The receiver or Processor has to ensure that the message is authentic and that the syntax and contents of the message are valid (done by automatic schema validation).
Initiating Actor	
Reacting Actor	Processor
Preconditions	
Postconditions	
Linked Use Cases (Extended By)	
Linked Use Cases (Includes)	Asynchronous ION use case processor side - Process ION request and send reply
Linked Use Cases (Realises)	
Base Activity	
Inputs	ION message : ION message with WSS
Outputs	delivery acknowledgement : deliveryAcknowledgement
Error Cases	delivery rejection : deliveryRejection

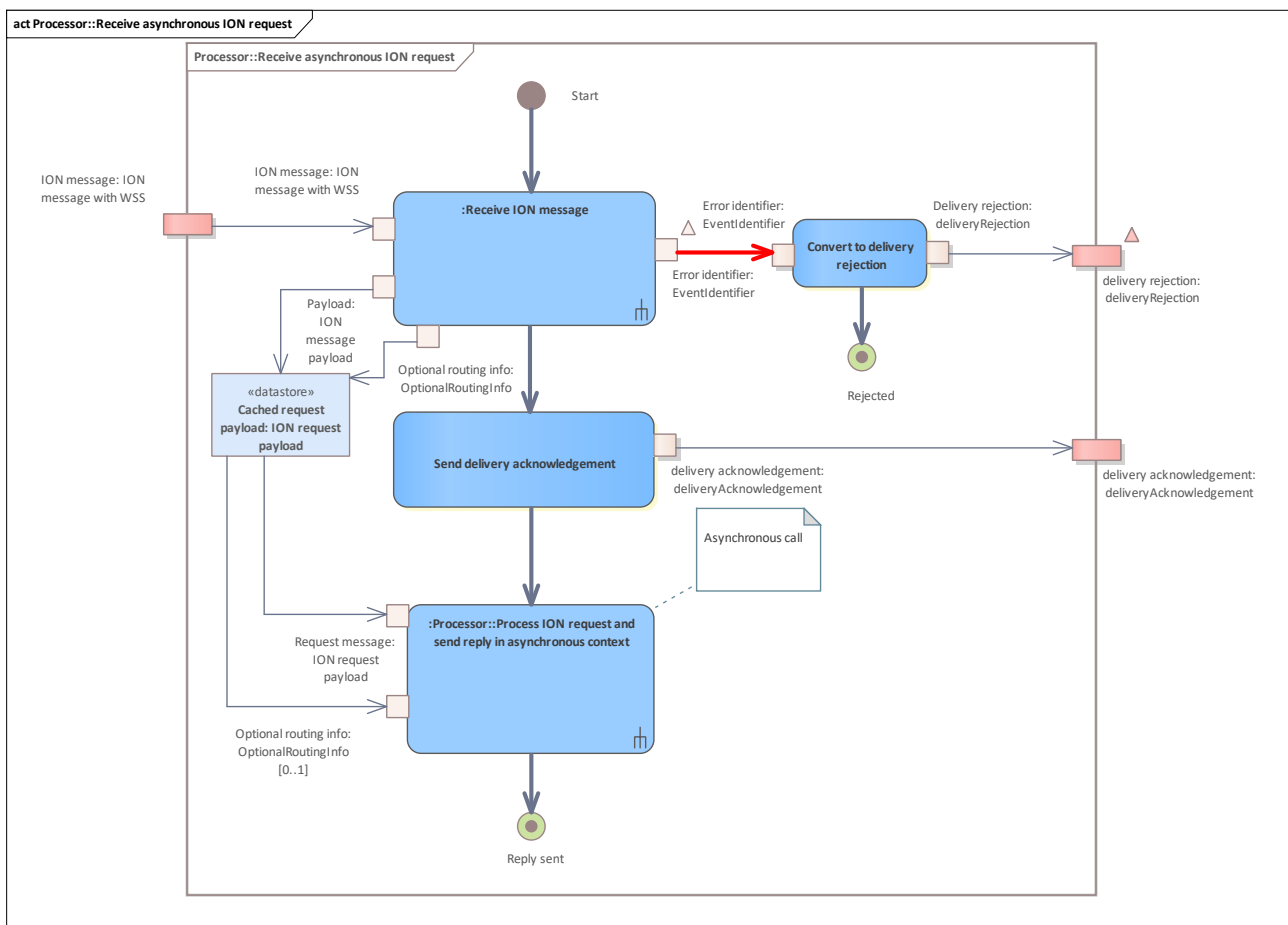
Activity Diagram

[Processor::Receive asynchronous ION request](#)

6.1.3.6.1 Processor::Receive asynchronous ION request

Activity to receive an ION request (see [Receive ION message](#)) in an asynchronous context. After the first message validation, a [deliveryAcknowledgement](#) is passed back to the [Initiator](#) if the underlying use case and communication are asynchronous.

6.1.3.6.1.1 Processor::Receive asynchronous ION request



Activity diagram for [Processor::Receive ION request in asynchronous context](#).

Receive ION message

See [Receive ION message](#).

Send delivery acknowledgement

Send a [DeliveryAcknowledgement](#) to

- the [Processor](#), if a [Request](#) has been received
- the [Initiator](#), if a [Reply](#) has been received

with a `<deliveredTo>RECIPIENT</deliveredTo>`.

Always done in an asynchronous context if the decryption and signature verification, the automatic schema validation and some basic checks were successful.

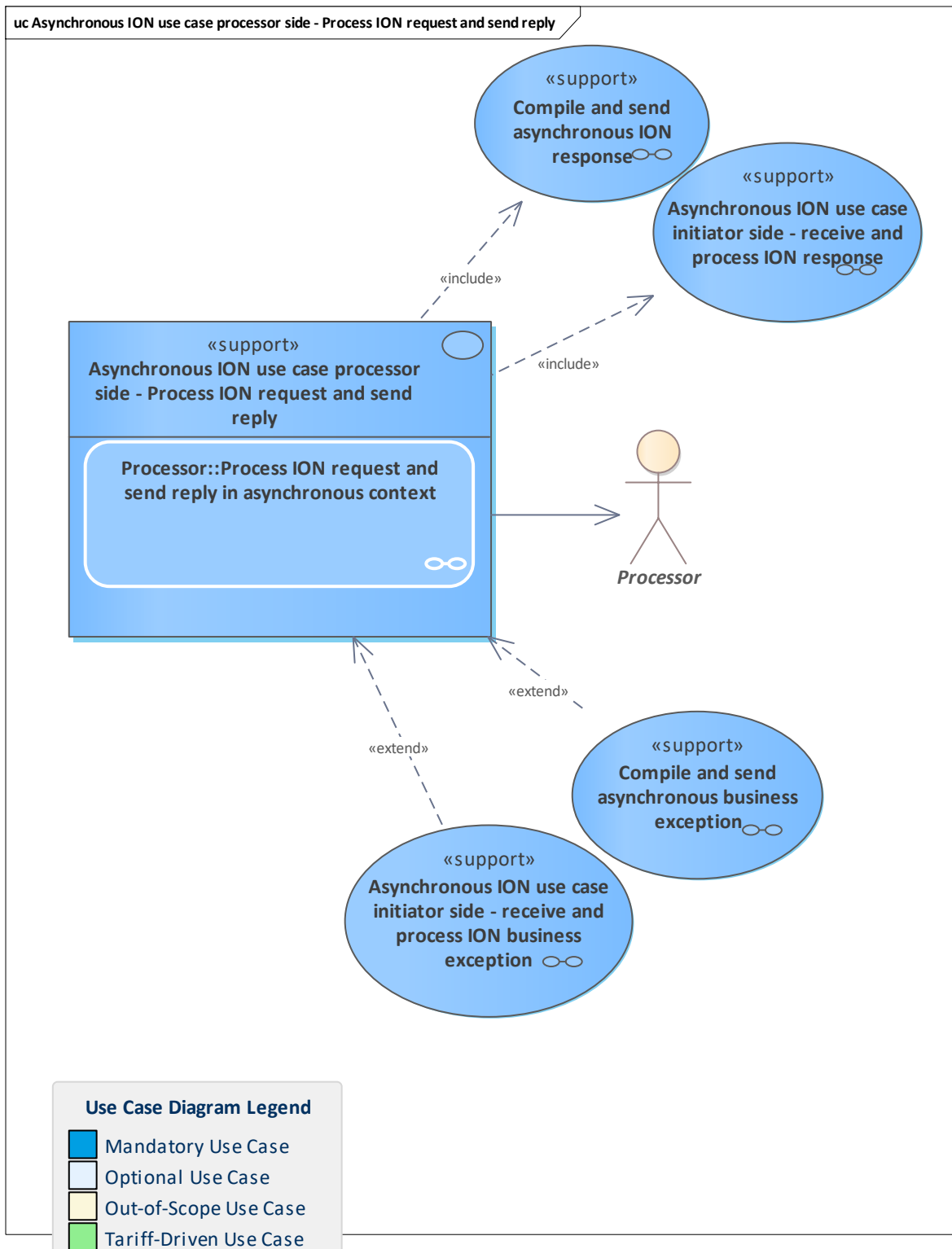
Processor::Process ION request and send reply in asynchronous context

See [Processor::Process ION request and send reply in asynchronous context](#).

Convert to delivery rejection

Embed the incoming [EventIdentifier](#) into a [deliveryRejection](#) with the specified format in [Asynchronous Reply Overview](#).

6.1.3.7 Asynchronous ION use case processor side - Process ION request and send reply



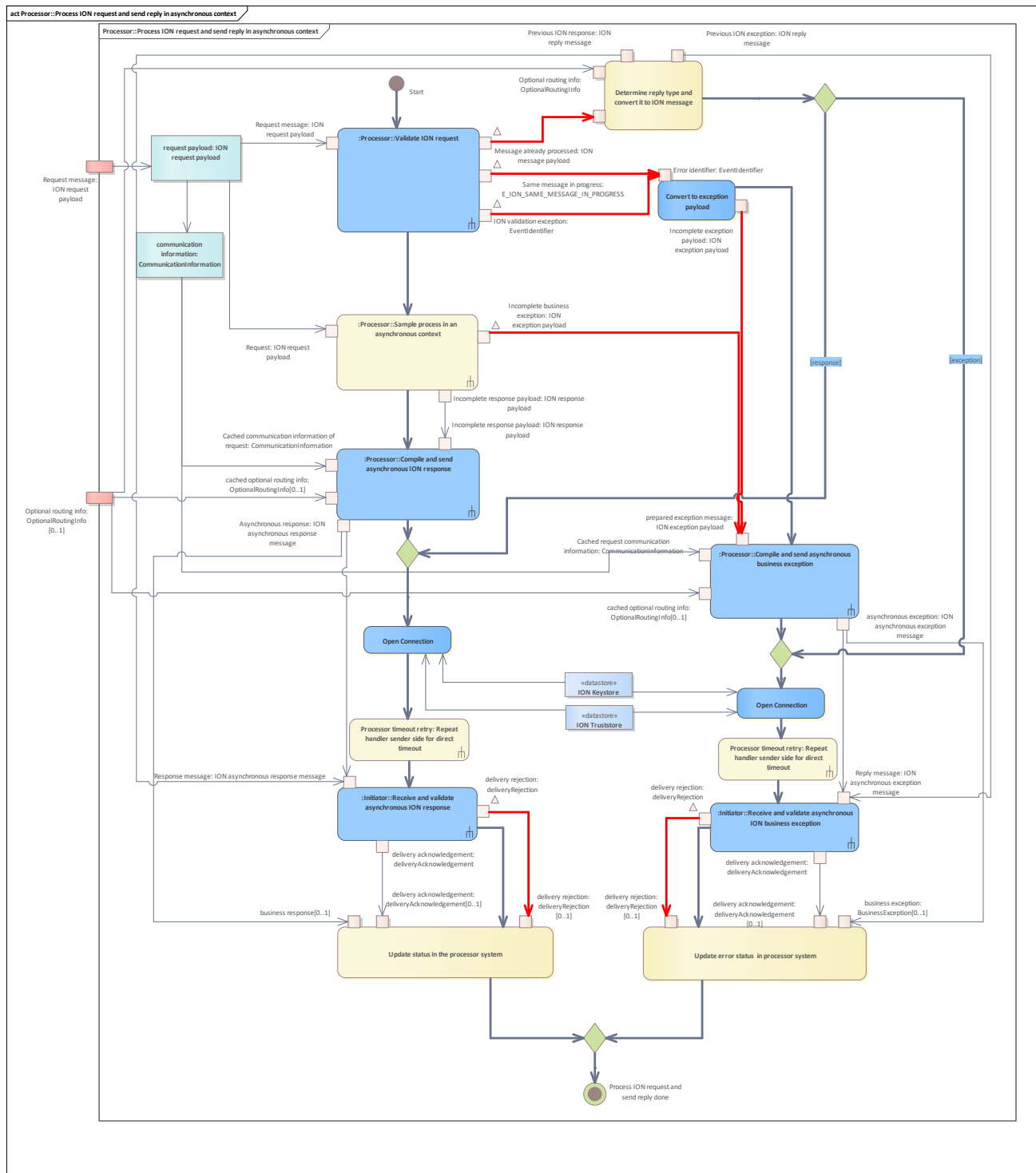
Use Case	Asynchronous ION use case processor side - Process ION request and send reply
Description	Use case that validates the ION request, calls the business part of the use case and sends a reply back to the Initiator . This scenario is valid for all derived use cases which have to process request messages in an asynchronous context. The ION message must not be a duplicate (except for certain Retry Handling). The ION message ID has to be unique. See Unique

	identifier for each ION message . The sender must be known in the system. The right sender role depends on the parent use case and has to be described there.
Initiating Actor	
Reacting Actor	Processor
Preconditions	
Postconditions	
Linked Use Cases (Extended By)	Asynchronous ION use case initiator side - receive and process ION business exception / Compile and send asynchronous business exception
Linked Use Cases (Includes)	Asynchronous ION use case initiator side - receive and process ION response / Compile and send asynchronous ION response
Linked Use Cases (Realises)	
Base Activity	
Inputs	Optional routing info : OptionalRoutingInfo Request message : ION request payload
Outputs	
Error Cases	
Activity Diagram	Processor::Process ION request and send reply in asynchronous context

6.1.3.7.1 Processor::Process ION request and send reply in asynchronous context

See [Asynchronous ION use case processor side - Process ION request and send reply](#).

6.1.3.7.1.1 Processor::Process ION request and send reply in asynchronous context



Activity diagram for [Asynchronous ION use case processor side - Process ION request and send reply](#).

Processor::Sample process in an asynchronous context

See [Processor::Sample process in an asynchronous context](#).

Processor::Compile and send asynchronous ION response

See [Processor::Compile and send asynchronous ION response](#).

Processor::Compile and send asynchronous business exception

See [Processor::Compile and send asynchronous business exception](#).

Initiator::Receive and validate asynchronous ION response

See [Initiator::Receive and validate asynchronous ION response](#).

Initiator::Receive and validate asynchronous ION business exception

See [Initiator::Receive and validate asynchronous ION business exception](#).

Determine reply type and convert it to ION message

Action that rebuilds an ION message for an existing [message payload](#) of a reply. Depending on the message type, either an [asynchronous ION response message](#) or an [asynchronous ION exception message](#) will be built and passed back in order to be directly sent to the sender.

Processor::Validate ION request

See [Processor::Validate ION request](#).

Repeat handler sender side for direct timeout

See [Repeat handler sender side for direct timeout](#).

Update status in the processor system

Update the status in the processor's system concerning the processed request and the response sent to the initiator. The final status update is the same for synchronous or asynchronous contexts.

In the case of an incoming [deliveryRejection](#), consider a manual scenario.

Update error status in processor system

Update the error status registering the occurred error and the reference to the sent exception. In the case of an incoming [deliveryRejection](#), consider a manual scenario.

Repeat handler sender side for direct timeout

See [Repeat handler sender side for direct timeout](#).

Convert to exception payload

Take an incoming [EventIdentifier](#), create the matching exception element including the correlation information of the original request without certain fields ([IonMessageId](#), etc.) of the [CommunicationInformation](#) which have to be filled later when the exception is sent.

Open Connection

Opens a TLS connection (to the CRE) presenting the [ION TLS certificate](#) . In case the CRE URL uses the HTTP scheme, open TCP connection or re-use an existing one.

Opens a TLS connection to the CRE presenting the [ION TLS certificate](#) found in the [ION Keystore](#) (see also [Keys and certificates : ION Key- and Truststore](#)) for the configured organisation ID. Checks the validity of the retrieved certificate against the [ION Truststore](#) (incorporating CRLs or OCSP). Asserts that the organisation ID given in the "o" part of the certificate subject matches 5602 (organisation ID of the CRE) and that there is no "ou" part (i.e. it is an TLS certificate).

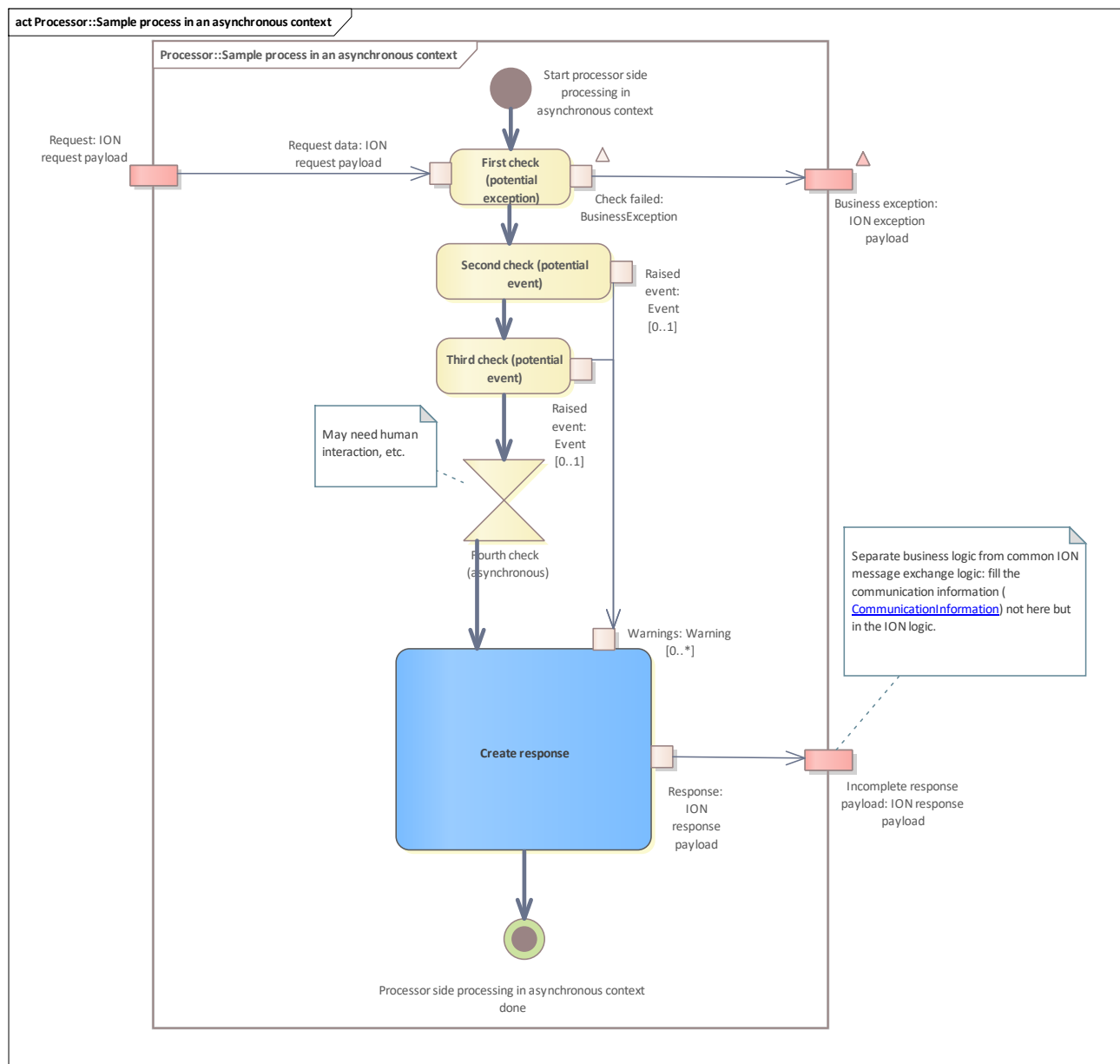
Open Connection

See [Open Connection](#)

6.1.3.7.1.2 Processor::Sample process in an asynchronous context

Activity that shows sample steps and checks on the server-side in an asynchronous context. This activity is embedded between [Receive and validate ION request](#) and [Compile and send asynchronous ION response](#) or [Compile and send asynchronous business exception](#) and performs the processor-side business logic.

6.1.3.7.1.2.1 Processor::Sample process in an asynchronous context



Activity diagram that shows the control flow, object flow and potential exception flow in the activity [Sample process server side in an asynchronous context](#) triggered by the use case [Asynchronous ION use case processor side](#).

Create response

Template action to create a [Response payload](#) object.

Compile [Response business data](#) (if any beyond the [BusinessAcknowledgement](#)).

Build a top-level element for the response payload containing

- "empty" [OriginalRequestCommunicationInformation](#) and [CommunicationInformation](#) objects¹
- the [WarningList](#) with the processing events - if any
- the newly compiled [Response business data](#) - if any

¹To keep the same steps of filling the [OriginalRequestCommunicationInformation](#) and [CommunicationInformation](#) objects in the response payload away from the business logic, this is done later in [Processor::Compile and send an asynchronous ION response](#). Also do not consider the (new) response ION message ID and your own organisation ID and role. These are also done there.

First check (potential exception)

Example for a check which is done with the [Request business data](#). This example check may raise a [BusinessException](#).

Second check (potential event)

Example for a check which is done with the [Request business data](#). This example check may raise an [Event](#).

Third check (potential event)

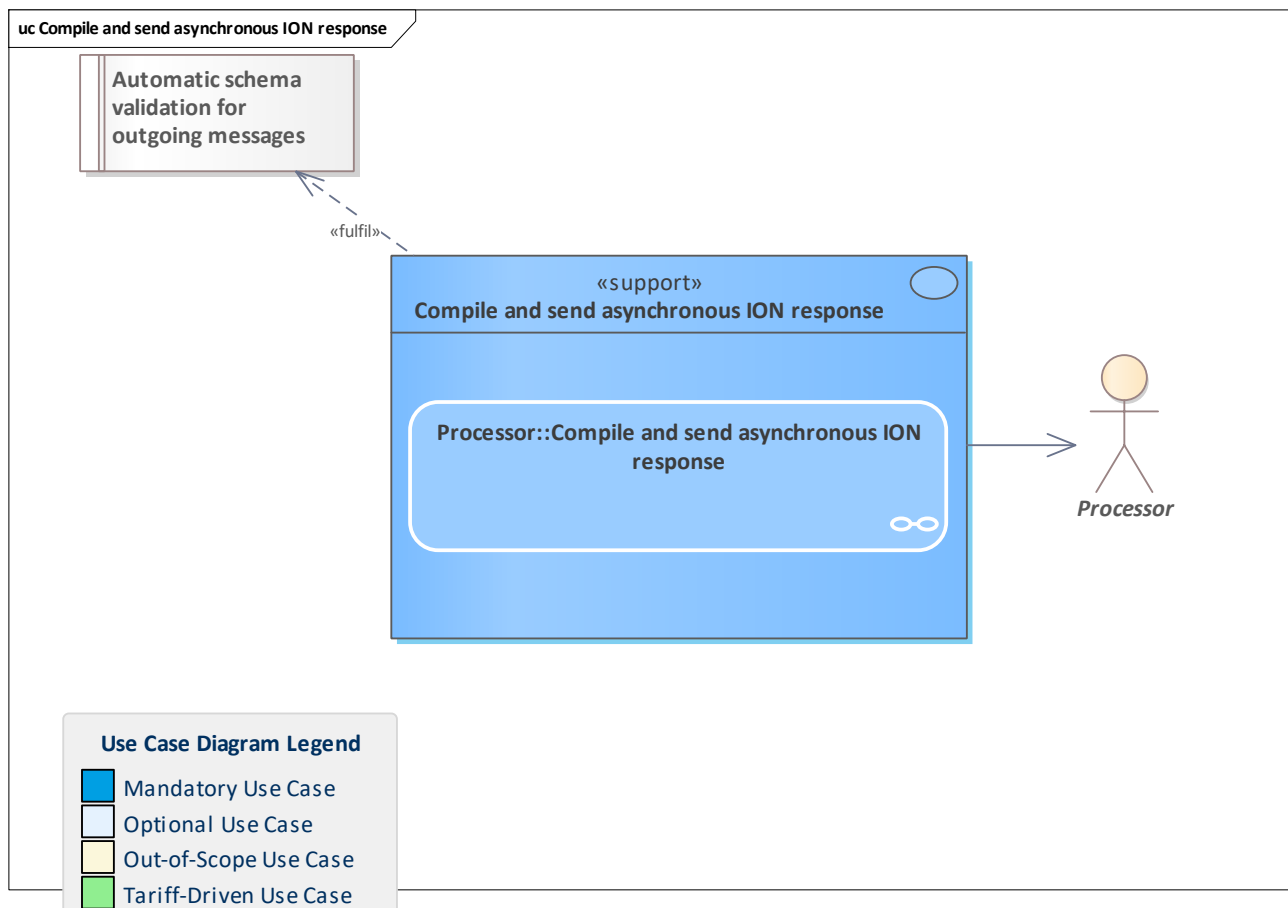
Example for a check which is done with the [Request business data](#). This example check may raise a further [Event](#).

Fourth check (asynchronous)

Action that is to show an asynchronous test step. For example, a system user activity may be required, a request to an external system or similar.

If such actions occur in an activity, the entire use case becomes asynchronous.

6.1.3.8 Compile and send asynchronous ION response



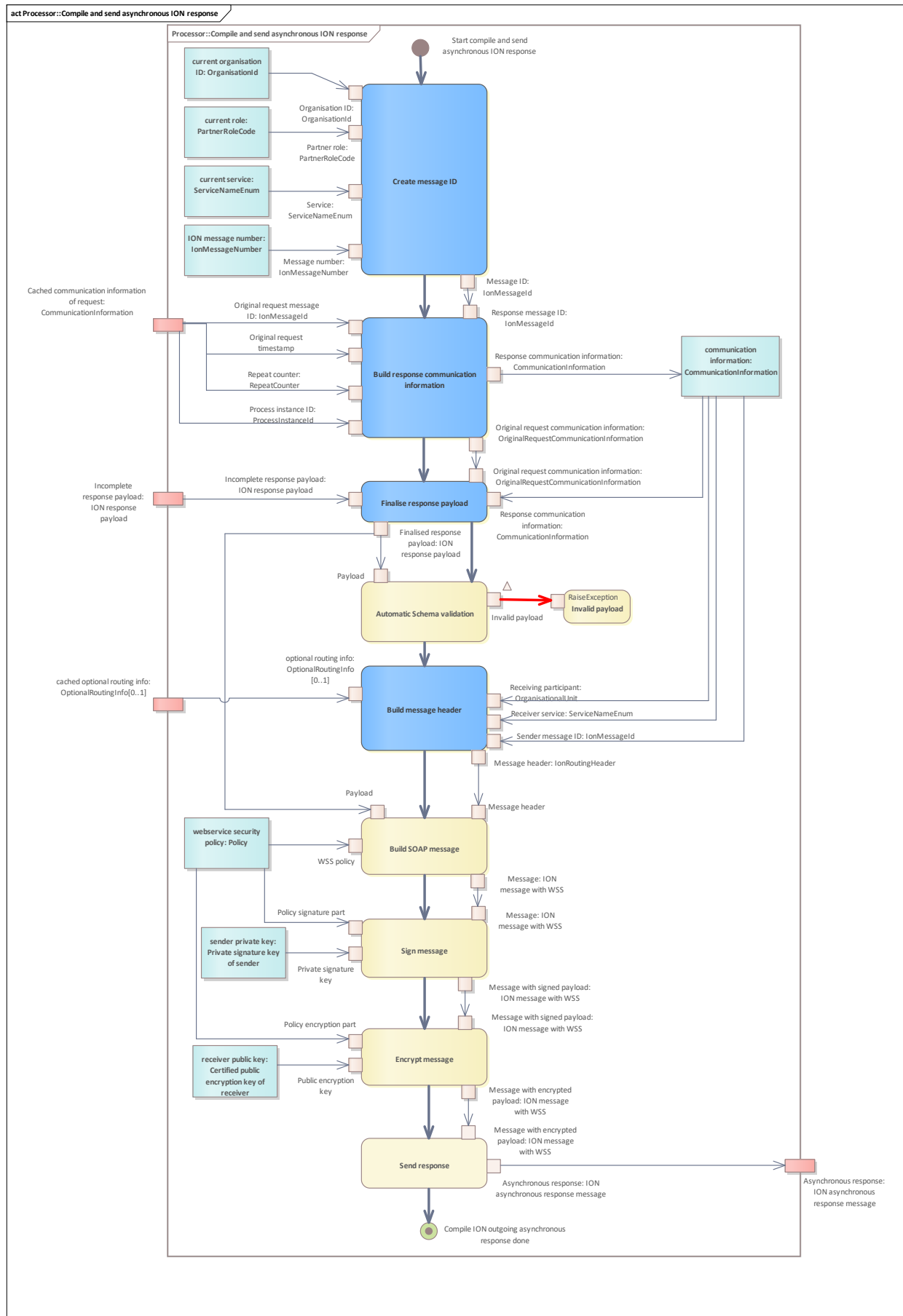
Use Case	Compile and send asynchronous ION response
Description	<p>The general use case for compiling and sending an ION message. To emphasize the business direction (Processor to Initiator), the message is called response.</p> <p>This supporting use case is performed by the Processor and called at the end of a Process processor side ION activity in an asynchronous context or Sample process processor side in an asynchronous context.</p> <p>The Processor has to ensure that the syntax and content of the message are valid (see Automatic schema validation for outgoing messages), that the message cannot be read by a third party (see ION:Privacy) and that the message is authentic (see ION:Authenticity).</p>
Initiating Actor	
Reacting Actor	Processor
Preconditions	
Postconditions	
Linked Use Cases (Extended By)	
Linked Use Cases (Includes)	
Linked Use Cases (Realises)	
Base Activity	
Inputs	cached optional routing info : OptionalRoutingInfo Cached communication information of request : CommunicationInformation

	Incomplete response payload : ION response payload
Outputs	Asynchronous response : ION asynchronous response message
Error Cases	
Activity Diagram	Processor::Compile and send asynchronous ION response

6.1.3.8.1 **Processor::Compile and send asynchronous ION response**

Activity for the use case [Compile and send asynchronous ION response](#).

6.1.3.8.1.1 Processor::Compile and send asynchronous ION response



Activity diagram for the use case [Compile and send asynchronous ION response](#) which shows the control flow, object flow and potential exception flow in the activity [Compile and send asynchronous ION response](#) triggered by the use case [Asynchronous ION use case processor side](#).

Encrypt message

Encrypt the message with the receiver's public key. This public key is certified and can be obtained from the PKI by its directory service.

Note: please consider the rules concerning the [Validity of Certificates](#).

Due to the web service security [Policy](#), the entire [SOAP Body](#) has to be encrypted. The [SOAP Header](#) remains in clear text for routing purposes. Furthermore, the signature of the [SOAP Body](#) also has to be encrypted. The encrypted signature data of the signed [SOAP Body](#) is stored in the [Security header](#) in the [SOAP Header](#) and is not part of the [SOAP Body](#).

Sign message

Sign the message parts as defined in the [Webservice Security Policy](#) with the private key for signature purposes (the corresponding public key is registered and certified in the PKI and can be accessed via the directory service of the PKI).

According to the web service security specification, the entire body of the SOAP message which contains the payload is signed.

Normally performed by the underlying WSS framework.

Automatic Schema validation

Do the automatic XML schema validation with the embedded payload.

Normally performed by the underlying web service framework.

Build SOAP message

Build the SOAP message employing a [SOAP Envelope](#) with consideration of the binding rules in the corresponding WSDL. These rules determine if an [IonRoutingHeader](#) has to be embedded in the [SOAP Header](#).

Furthermore, the embedded webservice security policy in the WSDL is used to sign and encrypt the necessary parts of the message.

Create message ID

Create a new [IonMessageId](#) which is embedded in the communication information of the payload.

Build response communication information

Complete the "existing" payload's communication information.

To build the [CommunicationInformation](#) object of the response:

- sender ID from request [IonMessageId](#) -> response receiver ID
- sender role from request [IonMessageId](#) -> response receiver role
- sender service from request [IonMessageId](#) -> response receiver service
- new [IonMessageId](#) for the response
- new message timestamp
- no [RepeatCounter](#)!

- request [ProcessInstanceId](#) -> response process instance ID

Take the following fields from [CommunicationInformation](#) of the original request and copy them to the response payload to build the [OriginalRequestCommunicationInformation](#):

- [IonMessageId](#)
- Timestamp
- [RepeatCounter](#) (if present)

Finalise response payload

Complete the "incomplete" response payload by adding the [OriginalRequestCommunicationInformation](#) object and the newly built [CommunicationInformation](#) object.

Build message header

Build the message header which is used in the SOAP message or [SOAP Envelope](#). This header is defined by [IonRoutingHeader](#) and will be embedded in the [SOAP Header](#).

The operation name needed in the [IonRoutingHeader](#) is to be derived from the request payload. Due to WS-I compatibility, this can be done very easily.

Example: Payload is called *notifyEntitlementIssued* -> operation name will be the same: *notifyEntitlementIssued*.

Furthermore, the interface version in the new [IonRoutingHeader](#) must be set to the one of the system that currently builds the message header.

In the asynchronous reply context, the [OptionalRoutingInfo](#) in the new [IonRoutingHeader](#) must be the same as in the original request of the [Initiator](#) (copy from request).

Invalid payload

Exception if an automatic schema validation error occurs.

Two scenarios may occur:

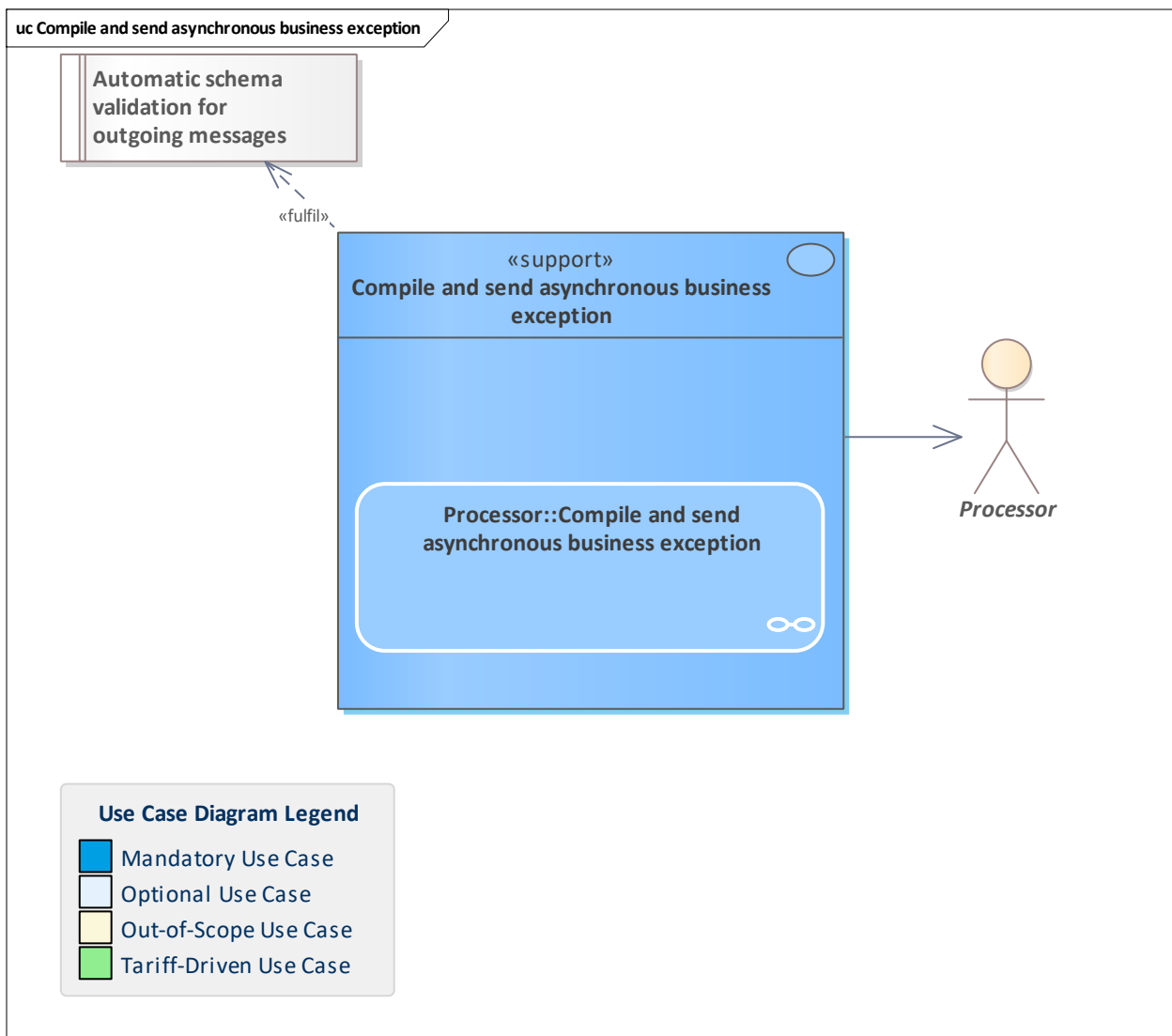
- If the automatic schema validation of the incoming message fails, the software implementation of the sender has to be fixed. The called operation terminates with this exception and no processing takes place
- If the automatic schema validation of the outgoing message fails, your software implementation or data storage has to be fixed. The process stops with this exception and prevents the case from above

For incoming messages, the underlying framework has to respect the requirement [Security relevant errors](#) to avoid DoS attacks. If the framework does not fulfil it, a separate exception handler must be employed.

Send response

Send the [asynchronous ION response message](#) with the signed and encrypted [Response payload](#).

6.1.3.9 Compile and send asynchronous business exception



Use Case	Compile and send asynchronous business exception
Description	<p>A general use case for compiling and sending an asynchronous ION exception message. This supporting use case is performed by the Processor and called at the end of a Process processor side ION activity in asynchronous context or Sample process processor side in an asynchronous context. The Processor has to ensure that the syntax and content of the message are valid (see Automatic schema validation for outgoing messages), that the message cannot be read by a third party (see ION:Privacy) and that the message is authentic (see ION:Authenticity).</p> <p>Compiles and sends a BusinessException named with a suitable use case-specific top- level element. This element is named after the operation provided by the Processor which was called by the Initiator, extended by the keyword "Exception". For example, notifyXY will be notifyXYException and this operation is provided by the Initiator.</p> <p>The exception can be considered as a response in the form of a BusinessException which has an additional Error to be evaluated.</p>

	<p>This error may occur in any step.</p> <p>The further processing - apart from the fact that the business process does not continue - in the direction of the ION is unchanged.</p> <p>Instead of the regular response, the exception is completed with CommunicationInformation and wrapped in top-level element, SOAP body and SOAP message with exactly the same steps.</p>
Initiating Actor	
Reacting Actor	Processor
Preconditions	
Postconditions	
Linked Use Cases (Extended By)	
Linked Use Cases (Includes)	
Linked Use Cases (Realises)	
Base Activity	
Inputs	cached optional routing info : OptionalRoutingInfo Incomplete exception payload : ION exception payload Cached request communication information : CommunicationInformation
Outputs	asynchronous exception : ION asynchronous exception message
Error Cases	
Activity Diagram	Processor::Compile and send asynchronous business exception

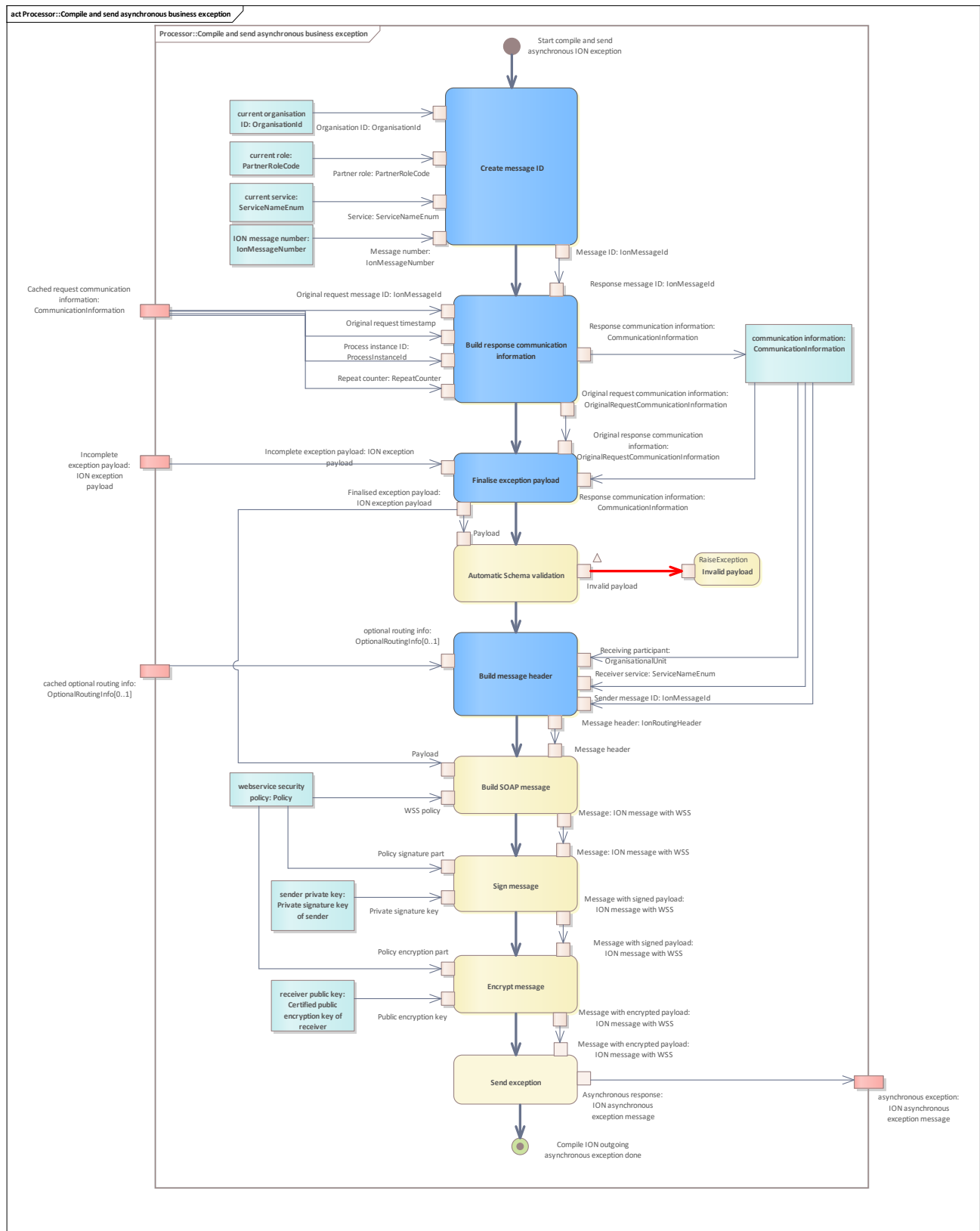
6.1.3.9.1 Processor::Compile and send asynchronous business exception

Activity which is a one-to-one copy of the [Processor::Compile and send asynchronous ION response](#) activity, except for

- the response payload from the business logic is not passed to the outgoing element
- the output type is an [asynchronous ION exception message](#) instead of an [asynchronous ION response message](#)

See also [Documentation: Compile and send asynchronous business exception](#).

6.1.3.9.1.1 Processor::Compile and send asynchronous business exception



Activity diagram for [Compile and send asynchronous business exception](#).

Create message ID

Create a new [IonMessageId](#) which is embedded in the communication information of the payload.

Build response communication information

Complete the "existing" payload's communication information.

To build the [CommunicationInformation](#) object of the response:

- sender ID from request [IonMessageId](#) -> response receiver ID
- sender role from request [IonMessageId](#) -> response receiver role
- sender service from request [IonMessageId](#) -> response receiver service
- new [IonMessageId](#) for the response
- new message timestamp
- no [RepeatCounter](#)!
- request [ProcessInstanceId](#) -> response process instance ID

Take the following fields from [CommunicationInformation](#) of the original request and copy them to the response payload to build the [OriginalRequestCommunicationInformation](#):

- [IonMessageId](#)
- Timestamp
- [RepeatCounter](#) (if present)

Finalise exception payload

Complete the "incomplete" [exception payload](#) by adding the [OriginalRequestCommunicationInformation](#) object and the newly built [CommunicationInformation](#) object.

Automatic Schema validation

Do the automatic XML schema validation with the embedded payload.
Normally performed by the underlying web service framework.

Invalid payload

Exception if an automatic schema validation error occurs.

Two scenarios may occur:

- If the automatic schema validation of the incoming message fails, the software implementation of the sender has to be fixed. The called operation terminates with this exception and no processing takes place
- If the automatic schema validation of the outgoing message fails, your software implementation or data storage has to be fixed. The process stops with this exception and prevents the case from above

For incoming messages, the underlying framework has to respect the requirement [Security relevant errors](#) to avoid DoS attacks. If the framework does not fulfil it, a separate exception handler must be employed.

Build message header

Build the message header which is used in the SOAP message or [SOAP Envelope](#). This header is defined by [IonRoutingHeader](#) and will be embedded in the [SOAP Header](#).

The operation name needed in the [IonRoutingHeader](#) is to be derived from the request payload. Due to WS-I compatibility, this can be done very easily.

Example: Payload is called *notifyEntitlementIssued* -> operation name will be the same: *notifyEntitlementIssued*.

Furthermore, the interface version in the new [IonRoutingHeader](#) must be set to the one of the system that currently builds the message header.

In the asynchronous reply context, the [OptionalRoutingInfo](#) in the new [IonRoutingHeader](#) must be the same as in the original request of the [Initiator](#) (copy from request).

Build SOAP message

Build the SOAP message employing a [SOAP Envelope](#) with consideration of the binding rules in the corresponding WSDL. These rules determine if an [IonRoutingHeader](#) has to be embedded in the [SOAP Header](#).

Furthermore, the embedded webservice security policy in the WSDL is used to sign and encrypt the necessary parts of the message.

Sign message

Sign the message parts as defined in the [Webservice Security Policy](#) with the private key for signature purposes (the corresponding public key is registered and certified in the PKI and can be accessed via the directory service of the PKI).

According to the web service security specification, the entire body of the SOAP message which contains the payload is signed.

Normally performed by the underlying WSS framework.

Encrypt message

Encrypt the message with the receiver's public key. This public key is certified and can be obtained from the PKI by its directory service.

Note: please consider the rules concerning the [Validity of Certificates](#).

Due to the web service security [Policy](#), the entire [SOAP Body](#) has to be encrypted. The [SOAP Header](#) remains in clear text for routing purposes. Furthermore, the signature of the [SOAP Body](#) also has to be encrypted. The encrypted signature data of the signed [SOAP Body](#) is stored in the [Security header](#) in the [SOAP Header](#) and is not part of the [SOAP Body](#).

Send exception

Send the [asynchronous ION exception message](#) with the signed and encrypted [exception payload](#).

6.1.3.10 Basic path with hidden CRE (explanatory purpose)

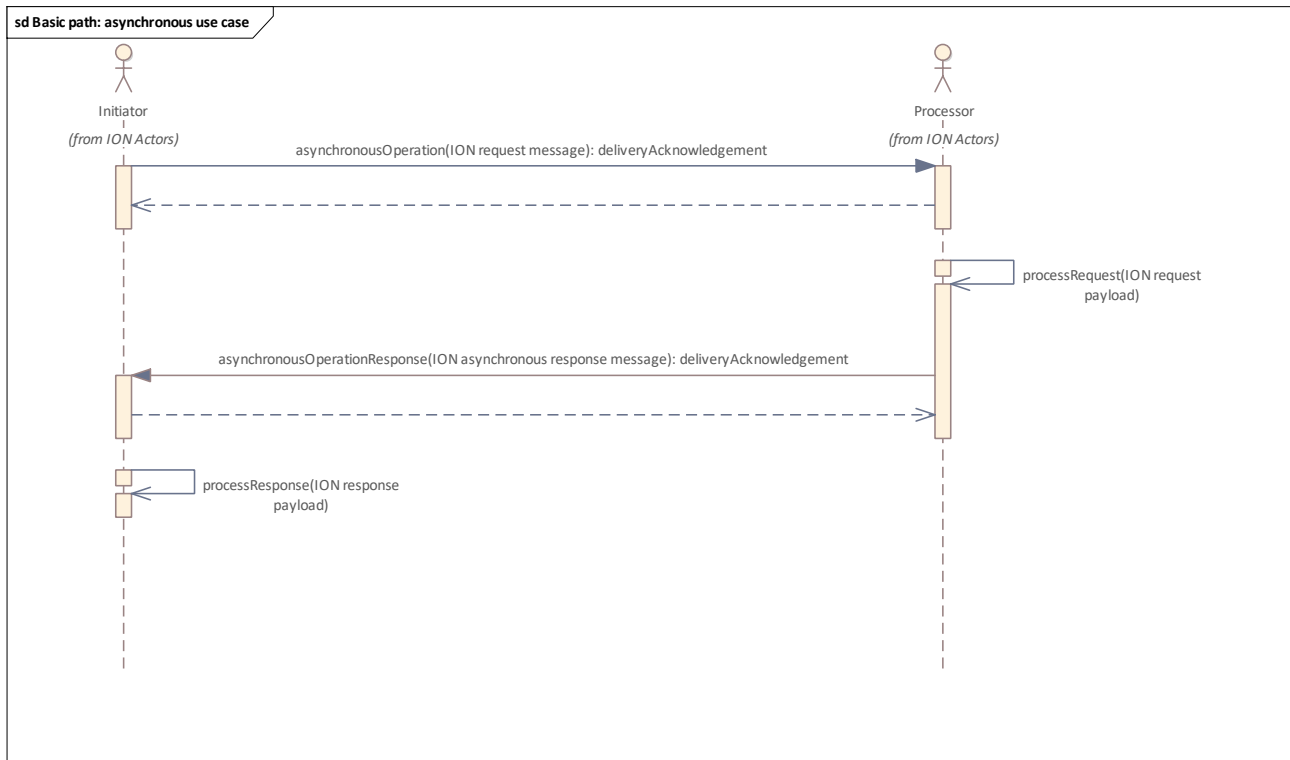
Sequence diagram which shows the basic flow of ION message exchange in an [Asynchronous ION use case](#).

The diagram shows an asynchronous message exchange scenario with two iterations.

- Iteration 1a: the [Initiator](#) starts with a new request message.
- Iteration 1b: the [Processor](#) takes this message, sends a technical reply ([DeliveryAcknowledgement](#)) and does its internal processing.
- Iteration 2a: after this step, for notification scenarios, a business acknowledgement ([BusinessAcknowledgement](#)) is sent. For get/mixed scenarios, an ION response with business data is sent.
- Iteration 2b: finally, after receiving the response message, a further [DeliveryAcknowledgement](#) is sent from the [Initiator](#) to the [Processor](#).

This diagram hides the intermediary [Central routing engine](#) to show only the pure message exchange between [Initiator](#) and [Processor](#).

6.1.3.10.1 Basic path: asynchronous use case



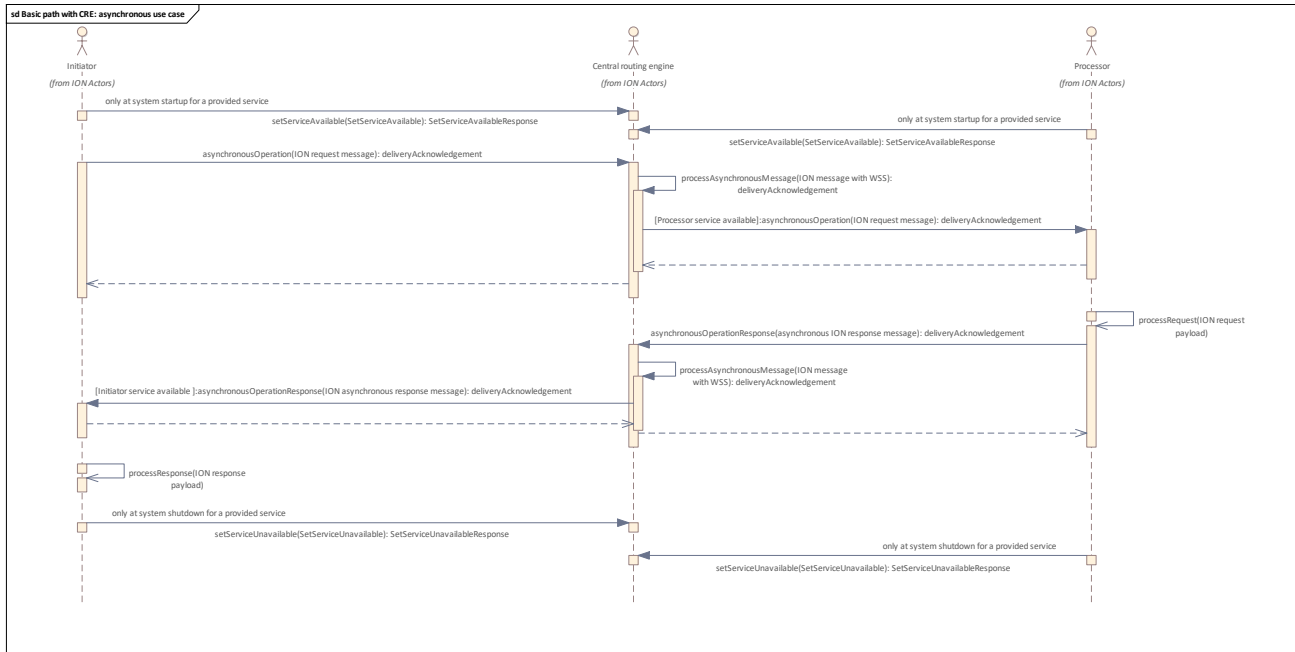
See [Basic path](#).

6.1.3.11 Basic path with CRE shown (real scenario)

Sequence diagram which shows the basic flow of ION message exchange in an [Asynchronous ION use case](#).

In addition to the interaction [Basic path](#), this diagram also shows the intermediary [Central routing engine](#) to display the real message exchange between [Initiator](#) and [Processor](#).

6.1.3.11.1 Basic path with CRE: asynchronous use case



See [Basic path with CRE](#).

6.1.3.12 Basic path with CRE: message store and forward (real scenario)

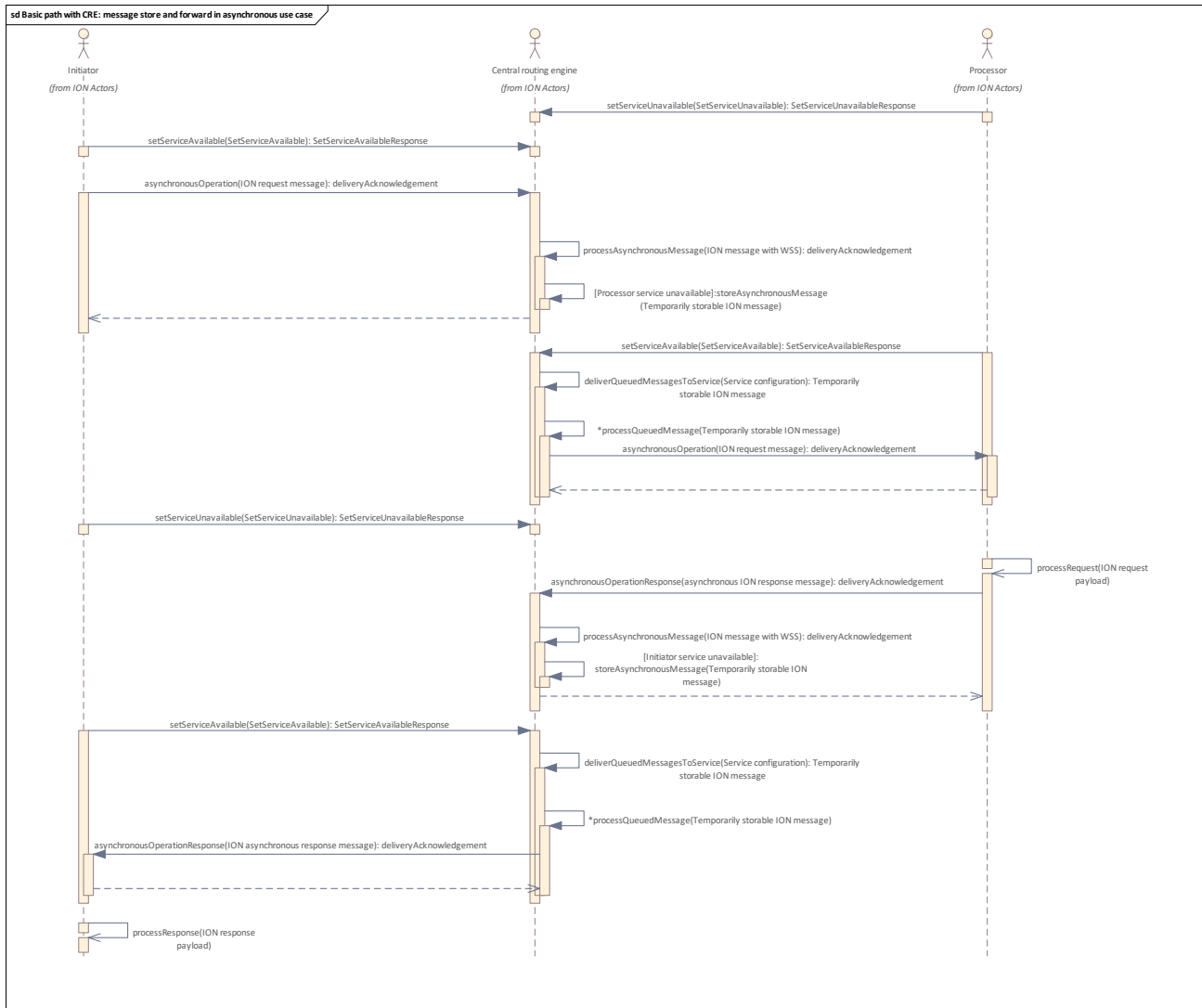
Sequence diagram which shows the basic flow of ION message exchange in an [Asynchronous ION use case](#).

This diagram also shows the intermediary [Central routing engine](#) to display the real message exchange between [Initiator](#) and [Processor](#).

Furthermore, a standard caching scenario is shown if either the initiator or the processor is not available (service is not available for recoverable reasons) to receive messages.

See also the use case for message caching in [Process message in asynchronous context with caching](#).

6.1.3.12.1 Basic path with CRE: message store and forward in asynchronous use case



See [Basic path with CRE: message caching](#).

6.1.3.13 Business exception path with hidden CRE (explanatory purpose)

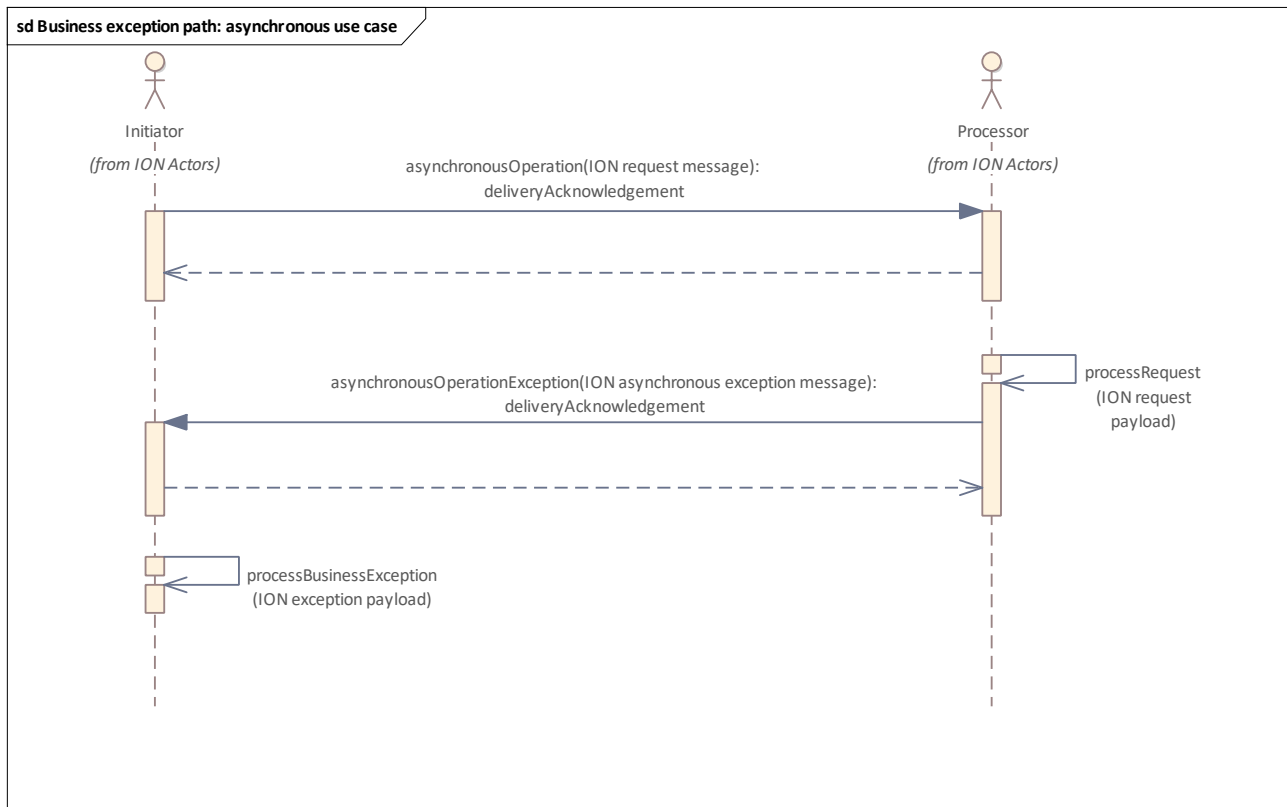
Sequence diagram which shows the business exception flow of ION message exchange in an [Asynchronous ION use case](#).

The diagram shows an asynchronous message exchange scenario with two iterations:

- Iteration 1a: the [Initiator](#) starts with a new request message.
- Iteration 1b: the [Processor](#) takes this message, sends a technical reply ([DeliveryAcknowledgement](#)) and starts its internal processing.
- Iteration 2a: while processing the request, a business exception occurs due to a failed check. Thus, the [Processor](#) sends a [BusinessException](#) to the [Initiator](#).
- Iteration 2b: finally, after receiving the business exception, a further [DeliveryAcknowledgement](#) is sent from the [Initiator](#) to the [Processor](#).

This diagram hides the intermediary [Central routing engine](#) to show only the pure message exchange between [Initiator](#) and [Processor](#).

6.1.3.13.1 Business exception path: asynchronous use case



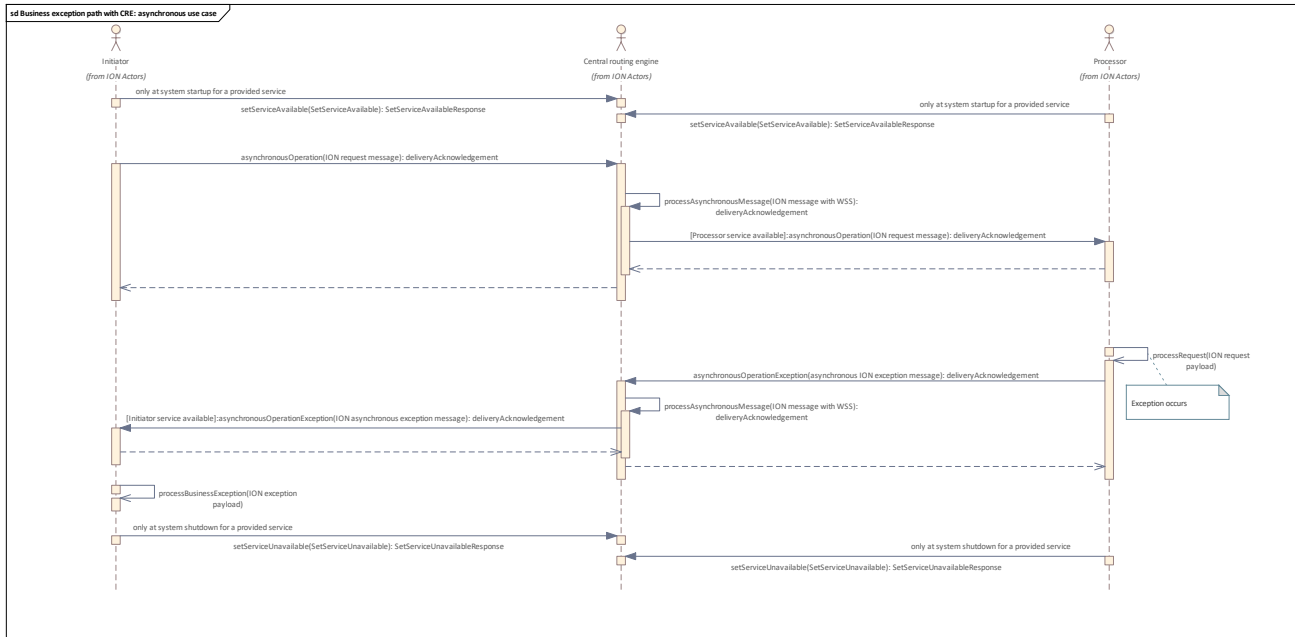
See [Business exception path](#).

6.1.3.14 Business exception path with CRE shown (real scenario)

Sequence diagram which shows the business exception flow of ION message exchange in an [Asynchronous ION use case](#).

In addition to the interaction [Business exception path](#), this diagram also shows the intermediary [Central routing engine](#) to display the real message exchange between [Initiator](#) and [Processor](#).

6.1.3.14.1 Business exception path with CRE: asynchronous use case



See [Business exception path with CRE](#).

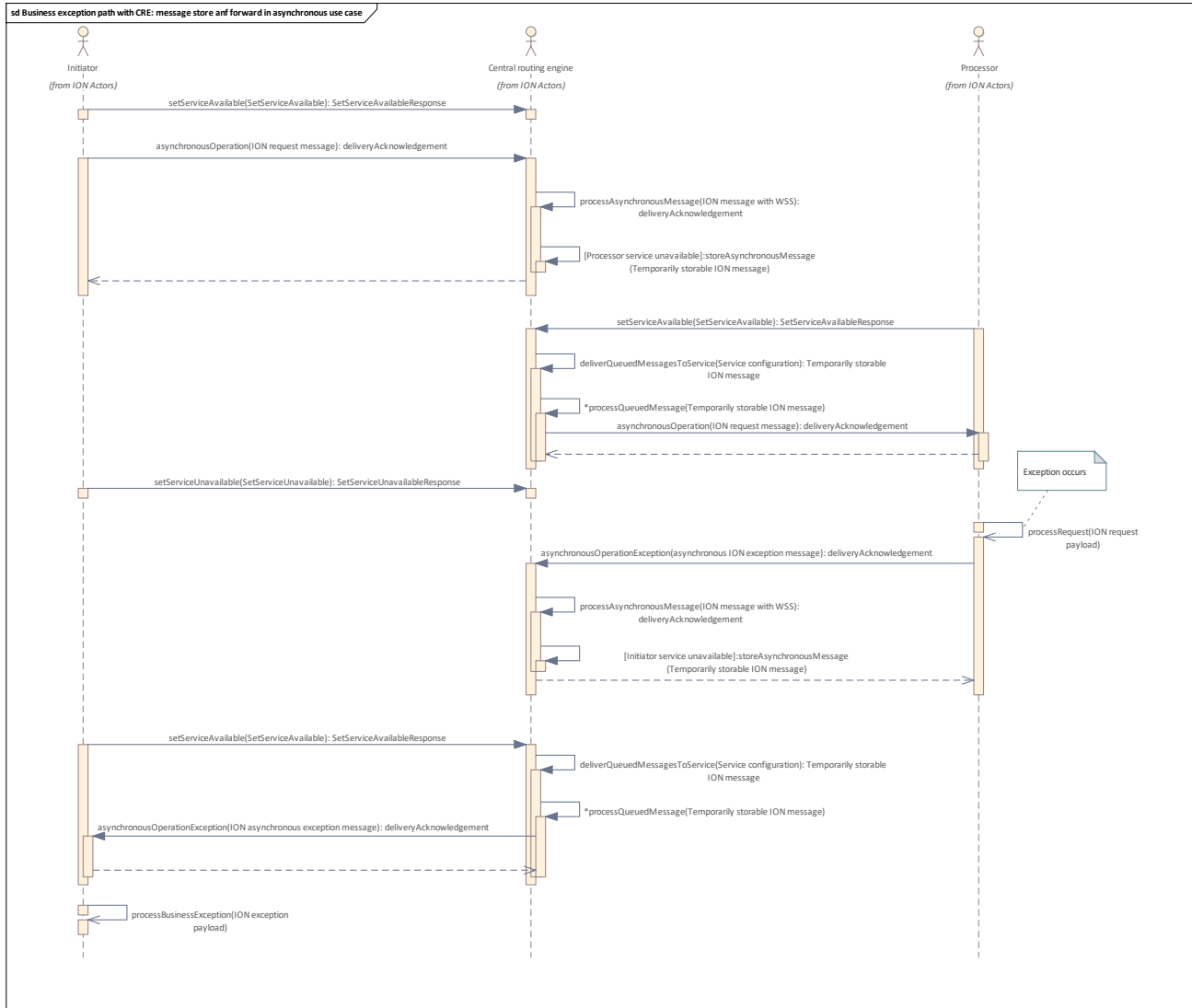
6.1.3.15 Business exception path with CRE: message store and forward (real scenario)

Sequence diagram which shows the basic flow of ION message exchange with a business exception in an [Asynchronous ION use case](#).

In addition to the interaction [Business exception path](#), this diagram also shows the intermediary [Central routing engine](#) to display the real message exchange between [Initiator](#) and [Processor](#). Furthermore, a standard caching scenario is shown if either the initiator or the processor or its needed service is not available to receive messages.

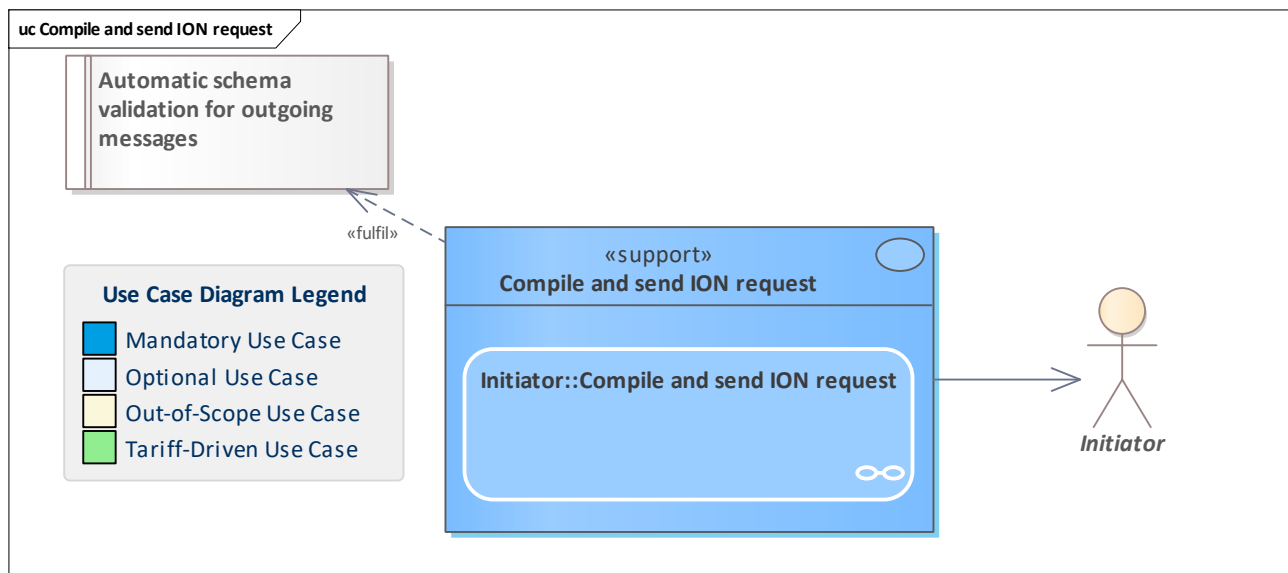
See also the use case for message caching in [Process message in asynchronous context with caching](#).

6.1.3.15.1 Business exception path with CRE: message store and forward in asynchronous use case



See [Business exception path with CRE: message caching](#).

6.1.4 Compile and send ION request



Use Case	Compile and send ION request
Description	<p>A common use case for compiling and sending an ION message. To emphasize the business direction (Initiator to Processor), the message is called <i>request</i>.</p> <p>This supporting use case is called inside of a Initiator::Sample process initiator side in an asynchronous context or Initiator::Sample process initiator side in a synchronous context. Thus, it can be used for synchronous and asynchronous parent use cases.</p> <p>The Initiator has to ensure that the syntax and content of the message are valid (see Automatic schema validation for outgoing messages), that the message cannot be read by a third party (see ION:Privacy) and that the message is authentic (see ION:Authenticity).</p> <p>The initiator should consider the lifecycle of its request or involved entity like "sent", "acknowledged", "in progress" or similar, depending on the concrete request, its intention and its data. For synchronous scenarios, the lifecycle mentioned above can be obsolete, since the response is directly available. The consideration of entity or message lifecycles is an implementation detail and is not part of this model/specification.</p>
Initiating Actor	
Reacting Actor	Initiator
Preconditions	
Postconditions	
Linked Use Cases (Extended By)	
Linked Use Cases (Includes)	
Linked Use Cases (Realises)	
Base Activity	
Inputs	Payload : ION request payload
Outputs	Outgoing request : ION request message

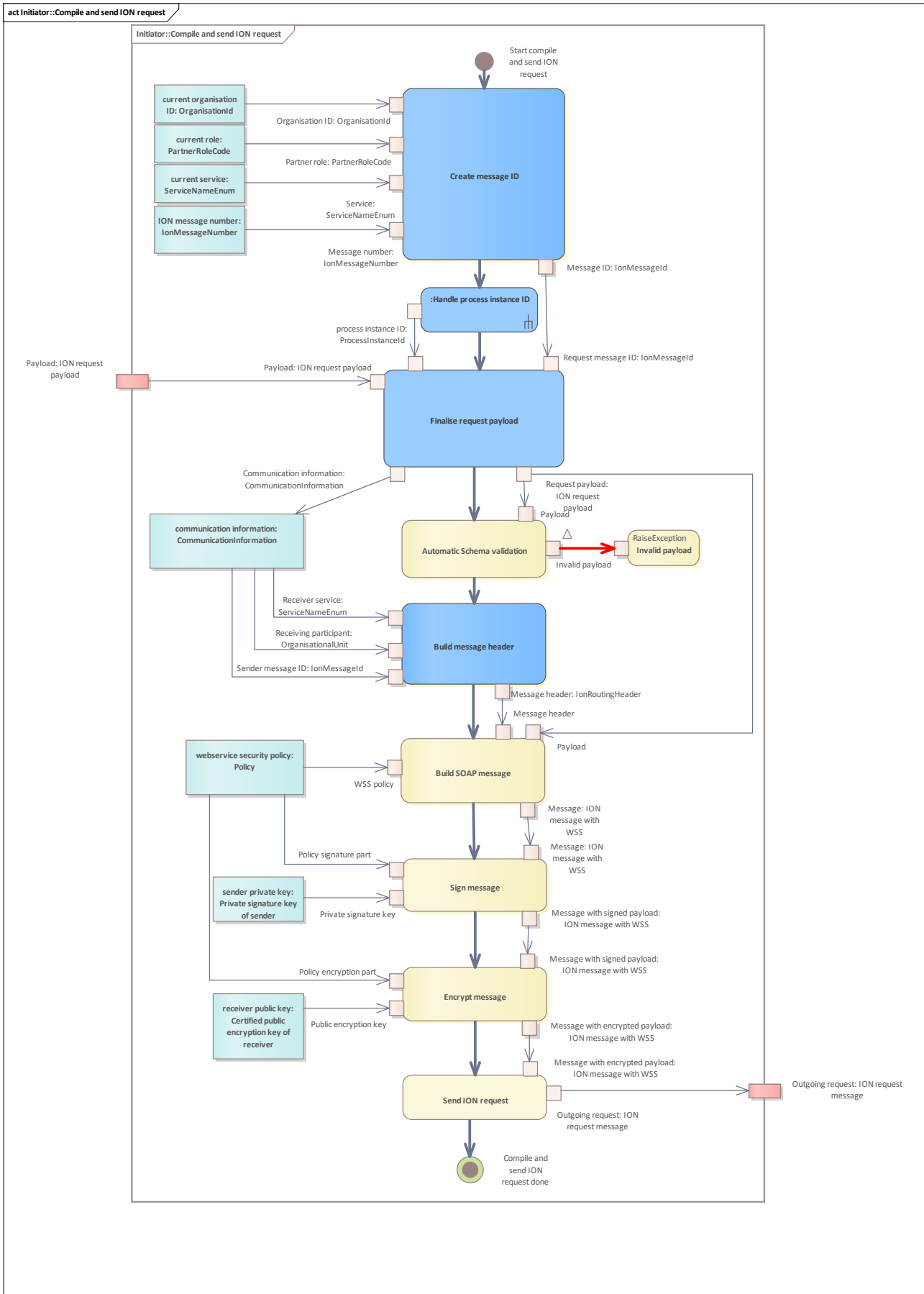
Error Cases	
Activity Diagram	Initiator::Compile and send ION request

6.1.4.1 Initiator::Compile and send ION request

Activity for supporting use case [Compile and send ION request](#).

The incoming [request payload](#) has been composed by the initiator's business logic as far as possible and will be finalized here (see diagram [ION Use Cases : Initiator::Compile and send ION request](#)).

6.1.4.1.1 Initiator::Compile and send ION request



Activity diagram for the use case [Compile and send ION request](#) which shows the control flow, object flow and potential exception flow in the activity [Compile and send ION request](#) triggered either by the use case [Asynchronous use case client side](#) or [Synchronous use case client side](#).

Create message ID

Create a new [IonMessageId](#) which is embedded in the communication information of the payload.

Finalise request payload

Set the created message ID in the request communication information of the request payload as well as the receiver role and service name (in case it was not yet set by business logic). If a role or service name has to be set, this information can be derived from the operation name.

Use a new timestamp ("now") for messageTimestamp in [CommunicationInformation](#).

For the [ProcessInstanceId](#) consider the following described in [Handle process instance ID](#).

Automatic Schema validation

Do the automatic XML schema validation with the embedded payload.
Normally performed by the underlying web service framework.

Invalid payload

Exception if an automatic schema validation error occurs.

Two scenarios may occur:

- If the automatic schema validation of the incoming message fails, the software implementation of the sender has to be fixed. The called operation terminates with this exception and no processing takes place
- If the automatic schema validation of the outgoing message fails, your software implementation or data storage has to be fixed. The process stops with this exception and prevents the case from above

For incoming messages, the underlying framework has to respect the requirement [Security relevant errors](#) to avoid DoS attacks. If the framework does not fulfil it, a separate exception handler must be employed.

Build message header

Build the message header which is used in the SOAP message or [SOAP Envelope](#). This header is defined by [IonRoutingHeader](#) and will be embedded in the [SOAP Header](#).

The operation name needed in the [IonRoutingHeader](#) is to be derived from the request payload. Due to WS-I compatibility, this can be done very easily.

Example: Payload is called *notifyEntitlementIssued* -> operation name will be the same: *notifyEntitlementIssued*.

Furthermore, the interface version in the new [IonRoutingHeader](#) must be set to the one of the system that currently builds the message header.

In the asynchronous reply context, the [OptionalRoutingInfo](#) in the new [IonRoutingHeader](#) must be the same as in the original request of the [Initiator](#) (copy from request).

Build SOAP message

Build the SOAP message employing a [SOAP Envelope](#) with consideration of the binding rules in the corresponding WSDL. These rules determine if an [IonRoutingHeader](#) has to be embedded in the [SOAP Header](#).

Furthermore, the embedded webservice security policy in the WSDL is used to sign and encrypt the necessary parts of the message.

Sign message

Sign the message parts as defined in the [Webservice Security Policy](#) with the private key for signature purposes (the corresponding public key is registered and certified in the PKI and can be accessed via the directory service of the PKI).

According to the web service security specification, the entire body of the SOAP message which contains the payload is signed.

Normally performed by the underlying WSS framework.

Encrypt message

Encrypt the message with the receiver's public key. This public key is certified and can be obtained from the PKI by its directory service.

Note: please consider the rules concerning the [Validity of Certificates](#).

Due to the web service security [Policy](#), the entire [SOAP Body](#) has to be encrypted. The [SOAP Header](#) remains in clear text for routing purposes. Furthermore, the signature of the [SOAP Body](#) also has to be encrypted. The encrypted signature data of the signed [SOAP Body](#) is stored in the [Security header](#) in the [SOAP Header](#) and is not part of the [SOAP Body](#).

Send ION request

Send the [ION request message](#) with the signed and encrypted [Request payload](#).

Handle process instance ID

See [Handle process instance ID](#).

6.1.5 Supporting activities

This package contains activities that are needed in more than one use case for ION messaging.

6.1.5.1 Supporting actions

This package contains actions that are needed in more than one use case for ION messaging.

6.1.5.1.1 Assert repeat counter is set and unknown

If the message ID was previously used, check if the repeat counter is set. The repeat counter belongs to the certain message ID and must be unique together with the message ID.

6.1.5.1.2 Register ION message ID (with repeat counter if any)

Register the current [IonMessageId](#). If a repeat counter was set, store it also as a tuple with the current message ID.

6.1.5.1.3 Check if operation supports message repetition

The WSDL documentation of the operation includes information regarding whether a specific operation must support message repetition.

The underlying concrete use case must correctly implement it.

In most cases, the repetition will not be needed in synchronous operations and will be needed in asynchronous operations.

6.1.5.1.4 Register repeat counter with current ION message ID

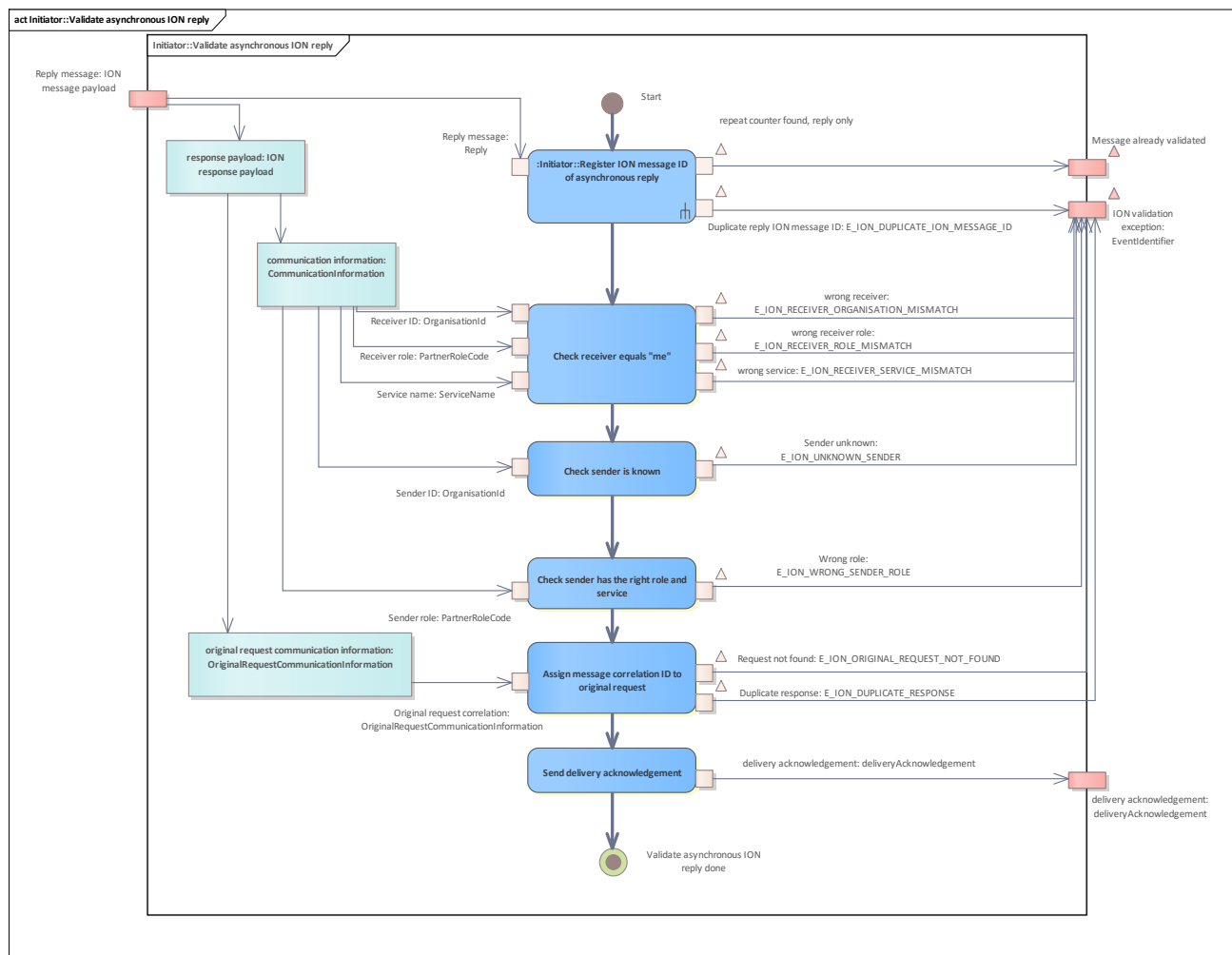
Register the repeat counter as a tuple with the current [IonMessageId](#) ([IonMessageId](#) and [RepeatCounter](#) - if set - together have to be unique).

6.1.5.2 Initiator::Validate asynchronous ION reply

Perform the more complex ION validation checks, including the registration of the incoming reply, taking into account the repeat counter.

See also [Receive and validate asynchronous ION response](#) and [Initiator::Receive and validate asynchronous ION business exception](#).

6.1.5.2.1 Initiator::Validate asynchronous ION reply



Activity diagram which shows the control flow, object flow and potential exception flow triggered by the activity [Receive and validate asynchronous ION response](#) and [Initiator::Receive and validate asynchronous ION business exception](#)

Initiator::Register ION message ID of asynchronous reply

See [Initiator::Register ION message ID of asynchronous reply](#).

Check receiver equals "me"

The receiver of the message must be my organisation ID and my role (see [PartnerRoleEnum](#)) and my service.

This check for organisation and role will never fail as long as the WSS is switched on and only becomes relevant without WSS.

The check for the service name may fail even if WSS is switched on since the service name is not part of the WSS parameters. A service name mismatch may only occur if a partner configured a wrong URL for its service.

Send delivery acknowledgement

Send a [DeliveryAcknowledgement](#) to

- the [Processor](#), if a [Request](#) has been received
- the [Initiator](#), if a [Reply](#) has been received

with a `<deliveredTo>RECIPIENT</deliveredTo>`.

Always done in an asynchronous context if the decryption and signature verification, the automatic schema validation and some basic checks were successful.

Check sender has the right role and service

Check if the sender has the right role and the right service to perform this use case and call the current operation.

This is important in the asynchronous scenario since the processor will try to reply to the sender role and service. If the sender service does not provide appropriate response / exception operations, replies can not be routed.

The list of services allowed to call an operation is part of the documentation of the operation in the WSDL.

Check sender is known

Check if the sender of the message exists in your own system's master data.

Note: each participating system is obliged to update its master data regularly by requesting the [Registrar](#) of the [Scheme Manager](#) for the organisation and role list of all participants.

Assign message correlation ID to original request

Check the correlation between the original request which caused the current response (also the current business exception, if the incoming answer is an exception) and this current incoming response by comparing the field "originalRequestCorrelationId" of the

[OriginalRequestCommunicationInformation](#) in the [BusinessAcknowledgement](#) or [BusinessException](#) with message IDs of the own request history.

Additionally, check if the current sender, sender role and sender service match the original receiver, receiver role and receiver service of the original request.

If no original request matches all checks from above, throw an

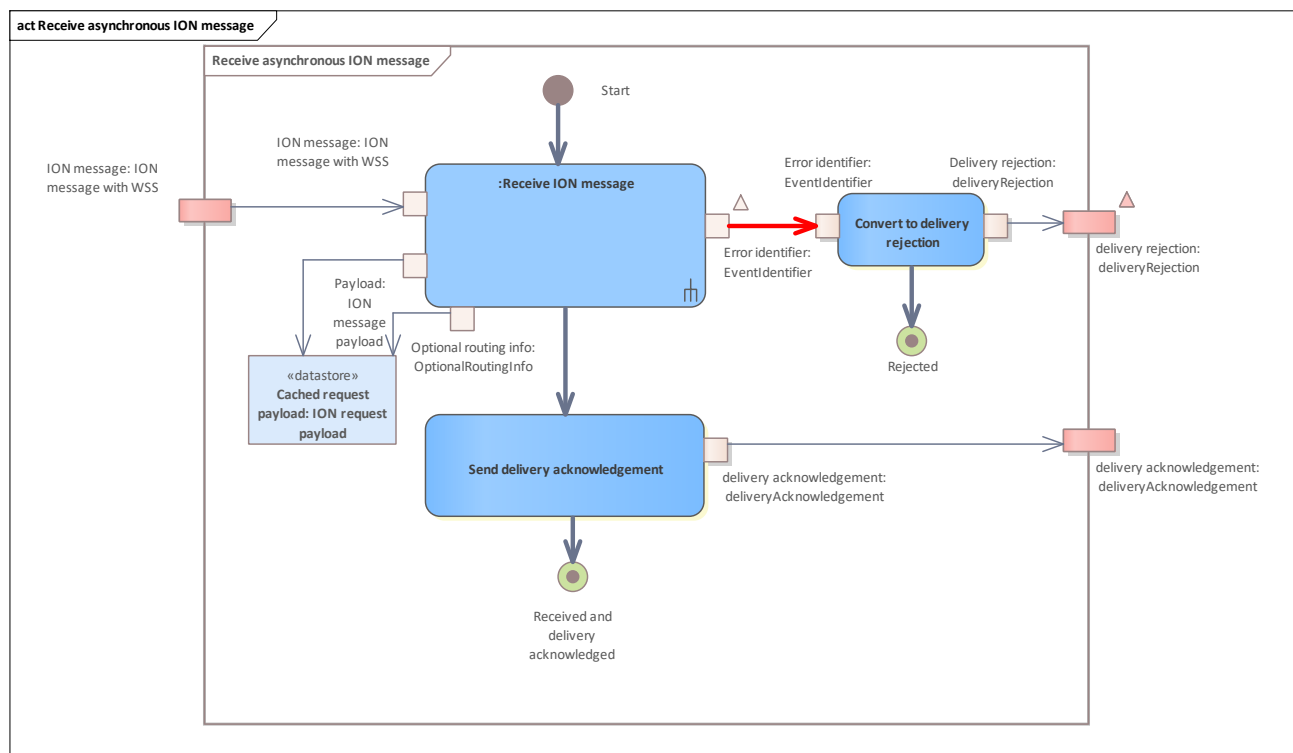
[E_IONC_ORIGINAL_REQUEST_NOT_FOUND](#) exception.

If a previous response exists, which is already correlated with the original request, throw an [E_IONC_DUPLICATE_RESPONSE](#) exception.

6.1.5.3 Receive asynchronous ION message

Activity to receive an ION message (see [Receive ION message](#)) in an asynchronous context. After the first message validation, a [deliveryAcknowledgement](#) is passed back to the [Initiator](#), since the underlying use case and the communication are assumed to be asynchronous.

6.1.5.3.1 Receive asynchronous ION message



Activity diagram for [Processor::Receive ION request in asynchronous context](#).

Send delivery acknowledgement

Send a [DeliveryAcknowledgement](#) to

- the [Processor](#), if a [Request](#) has been received
- the [Initiator](#), if a [Reply](#) has been received

with a `<deliveredTo>RECIPIENT</deliveredTo>`.

Always done in an asynchronous context if the decryption and signature verification, the automatic schema validation and some basic checks were successful.

Receive ION message

See [Receive ION message](#).

Convert to delivery rejection

Embed the incoming [EventIdentifier](#) into a [deliveryRejection](#) with the specified format in [Asynchronous Reply Overview](#).

6.1.5.4 Receive ION message

A common activity for receiving a message.

It does certain checks on incoming messages which have to be done for all parent use cases in the same way.

The receiver or [Processor](#) has to ensure that the message is authentic and that the syntax and contents of the message are valid (done by automatic schema validation).

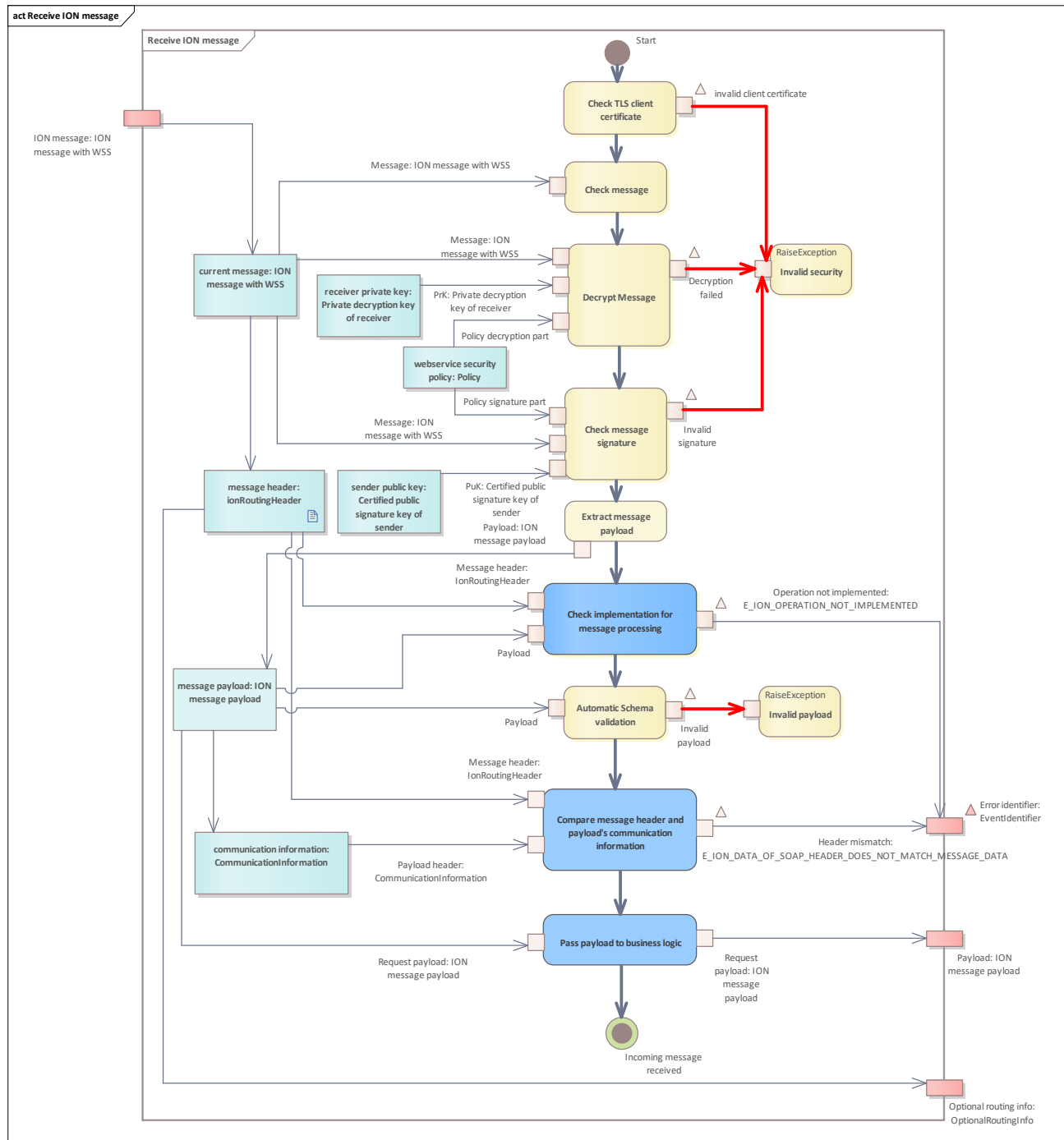
Besides decrypting the message and verifying its signature, the automatic schema validation ensures that a valid message is given to the business logic.

This activity can be used by all use cases which have to receive request messages. It can be used for synchronous and asynchronous parent use cases.

The performed steps are done for

- Receiving an [ION request message](#) (synchronous or asynchronous context)
- Receiving an [asynchronous ION response message](#)
- Receiving an [asynchronous ION exception message](#)

6.1.5.4.1 Receive ION message



Activity diagram for [Receive ION message](#).

Check implementation for message processing

Interceptor that performs a lookup for the operation that implements the processing of the incoming message. Due to the etiCORE release management, certain functionality may not be implemented, since it belongs to a higher minor version that is already implemented by the initiator but not by the processor.

The interceptor may work either with the header data or the payload. This depends on the position of the interceptor in the processing chain and is not determined here.

This action must be done before the schema validation runs.

Check message

Check if the SOAP message is well-formed and complies with the standard SOAP schema and the binding defined in the WSDL:

- SOAP message in general
- SOAP header composition
- WS-policy

Normally performed by the underlying framework.

Extract message payload

Normally done by the underlying web service framework. The payload is the etiCORE defined embedded top-level element in the [SOAP Body](#) contained in the [Encrypted payload](#). The payload contains optionally the [Request business data](#) (here: abstract placeholder for all implementing business data request classes).

Invalid payload

Exception if an automatic schema validation error occurs.

Two scenarios may occur:

- If the automatic schema validation of the incoming message fails, the software implementation of the sender has to be fixed. The called operation terminates with this exception and no processing takes place
- If the automatic schema validation of the outgoing message fails, your software implementation or data storage has to be fixed. The process stops with this exception and prevents the case from above

For incoming messages, the underlying framework has to respect the requirement [Security relevant errors](#) to avoid DoS attacks. If the framework does not fulfil it, a separate exception handler must be employed.

Invalid security

Exception if the message cannot be decrypted or that the message's signature cannot be verified.

The most commonly used error message in the different WSS frameworks is "Invalid security header".

In most of these cases, a wrong key reference is sent, the normalisation of the message is wrong or the message does not meet the requirements of the etiCORE WS policy (see [Appendix: etiCORE Standard-Policy](#)).

The underlying framework has to respect the requirement [Security relevant errors](#). If the framework does not fulfil it, a separate exception handler must be employed.

Automatic Schema validation

Do the automatic XML schema validation with the embedded payload.

Normally performed by the underlying web service framework.

Check message signature

Check the compliance to the WSS policy described in the chapter [Message-based encryption \(application layer\)](#) and [Appendix: etiCORE Standard-Policy](#).

Check the signature with the right certified public key from the directory service of the PKI. Identify the certificate matching all of the following parameters

- the organisation ID of the sender (with certificate subject)
- the role of the sender (with certificate subject)
- the purpose ("sig"), see [Basic Requirements](#)
- the serial number of the key located in the security header

Furthermore, consider the rules concerning the [Validity of Certificates](#).

Check that the signature was performed by the sender and its role is located in the SOAP Header.

Partially performed by the underlying web service framework's security provider.

Note: due to the etiCORE [Policy](#), only the signature key's serial number is contained in the [Security header](#) of the [SOAP header](#) in clear text.

Note: consider the [Transition Phase for Certificate Renewal](#) and [Security Levels](#).

Decrypt Message

Check the compliance to the WSS policy described in the chapter [Message-based encryption \(application layer\)](#) and [Appendix: etiCORE Standard-Policy](#).

Decrypt the incoming SOAP message using your own private key for

- the organisation
- the role
- the purpose (here: decryption)

Only the SOAP body is encrypted. The encryption information is located in the security header embedded in the SOAP header.

Normally performed by the underlying web service framework's security provider.

Note: consider the [Transition Phase for Certificate Renewal](#) and [Security Levels](#).

Compare message header and payload's communication information

Compare the fields of the [IONRoutingHeader](#) with the corresponding fields in the [CommunicationInformation](#) of the [request payload](#):

- [IONRoutingHeader](#).senderId -> [CommunicationInformation](#).messageId.senderId
- [IONRoutingHeader](#).senderRole -> [CommunicationInformation](#).messageId.senderRole
- [IONRoutingHeader](#).senderService -> [CommunicationInformation](#).messageId.senderService
- [IONRoutingHeader](#).messageNumber -> [CommunicationInformation](#).messageId.messageNumber
- [IONRoutingHeader](#).receiverId -> [CommunicationInformation](#).receiverId
- [IONRoutingHeader](#).receiverRole -> [CommunicationInformation](#).receiverRole
- [IONRoutingHeader](#).receiverService -> [CommunicationInformation](#).receiverService
- [IONRoutingHeader](#).repeatCounter -> [CommunicationInformation](#).repeatCounter
- [IONRoutingHeader](#).processInstanceId -> [CommunicationInformation](#).processInstanceId
- [IONRoutingHeader](#).operationName -> name of the payload element (without namespace)

Pass payload to business logic

Allow the payload to be returned to the higher-level business logic after the common checks are done.

Check TLS client certificate

The sender has to provide its TLS certificate.

All certificates for ION message exchange are issued by one defined authority and have a certain validity.

To verify the client certificate, an OSCP call has to be made to the trust centre or a current certificate revocation list has to be evaluated. Furthermore, the complete certificate chain has to be evaluated. Each certificate of the chain must be valid.

Normally, this is performed in the upstream web server of the receiving system.

See also [Transport encryption via TLS](#) and [Validity of Certificates](#).

Additionally, this check has to ensure that the right security level is used, see also [Security Levels](#).

Note: If the receiving party is a participant system or a JSB, the TLS client certificate should be always the one of the CRE. If the receiving party is the CRE, it has to verify the TLS client certificate of the participant system or the JSB.

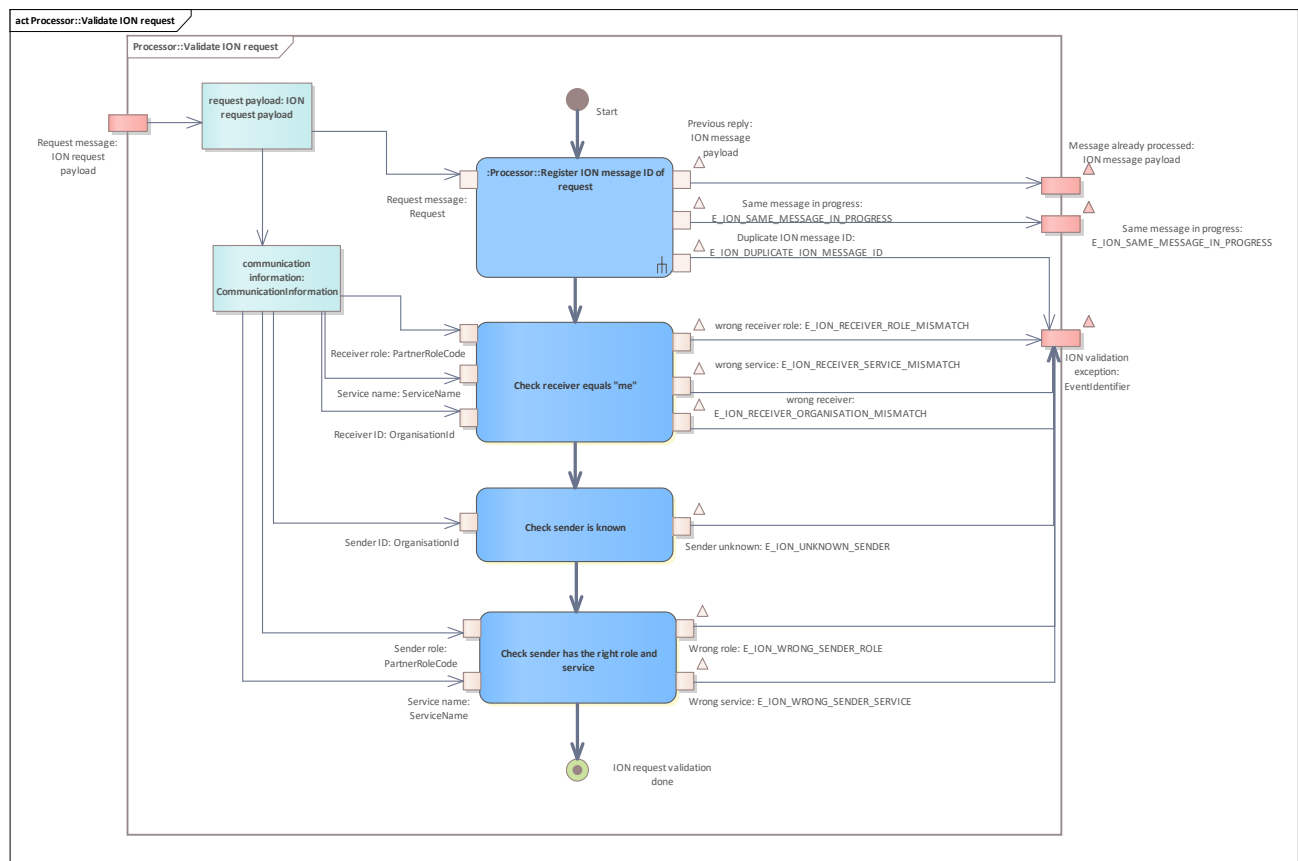
6.1.5.5 Processor::Validate ION request

Activity to validate an incoming ION request on the [Processor](#) side.

Normally called after the activity [Receive ION message](#).

This validation includes the registration of the request and the data validation of [CommunicationInformation](#).

6.1.5.5.1 Processor::Validate ION request



Activity diagram for [Processor::Validate ION request](#).

Check receiver equals "me"

The receiver of the message must be my organisation ID and my role (see [PartnerRoleEnum](#)) and my service.

This check for organisation and role will never fail as long as the WSS is switched on and only becomes relevant without WSS.

The check for the service name may fail even if WSS is switched on since the service name is not part of the WSS parameters. A service name mismatch may only occur if a partner configured a wrong URL for its service.

Check sender has the right role and service

Check if the sender has the right role and the right service to perform this use case and call the current operation.

This is important in the asynchronous scenario since the processor will try to reply to the sender role and service. If the sender service does not provide appropriate response / exception operations, replies can not be routed.

The list of services allowed to call an operation is part of the documentation of the operation in the WSDL.

Check sender is known

Check if the sender of the message exists in your own system's master data.

Note: each participating system is obliged to update its master data regularly by requesting the [Registrar](#) of the [Scheme Manager](#) for the organisation and role list of all participants.

Processor::Register ION message ID of request

See [Processor::Register ION message ID of request](#).

6.1.5.6 Processor::Register ION message ID of request

In the ION, each message has to be uniquely identified by a unique (composed) ID (see requirement [Unique identifier for each ION message](#)). If the system receives a request/response/exception message with an ID that already exists in the system, in most of the cases the message has to be refused. Then, the [IonMessageId](#) is not registered again and a (re-)processing of the message is rejected.

In the request context, an [E ION DUPLICATE ION MESSAGE ID](#) will be embedded in a [BusinessException](#).

The [IonMessageId](#) is also registered if a business exception is raised in the downstream business logic. This prevents having messages with the same ID, on one hand with a business acknowledgement and on the other hand with a business exception, as a response.

Thus, the message producer has to use a new ION message ID if a business exception has been received.

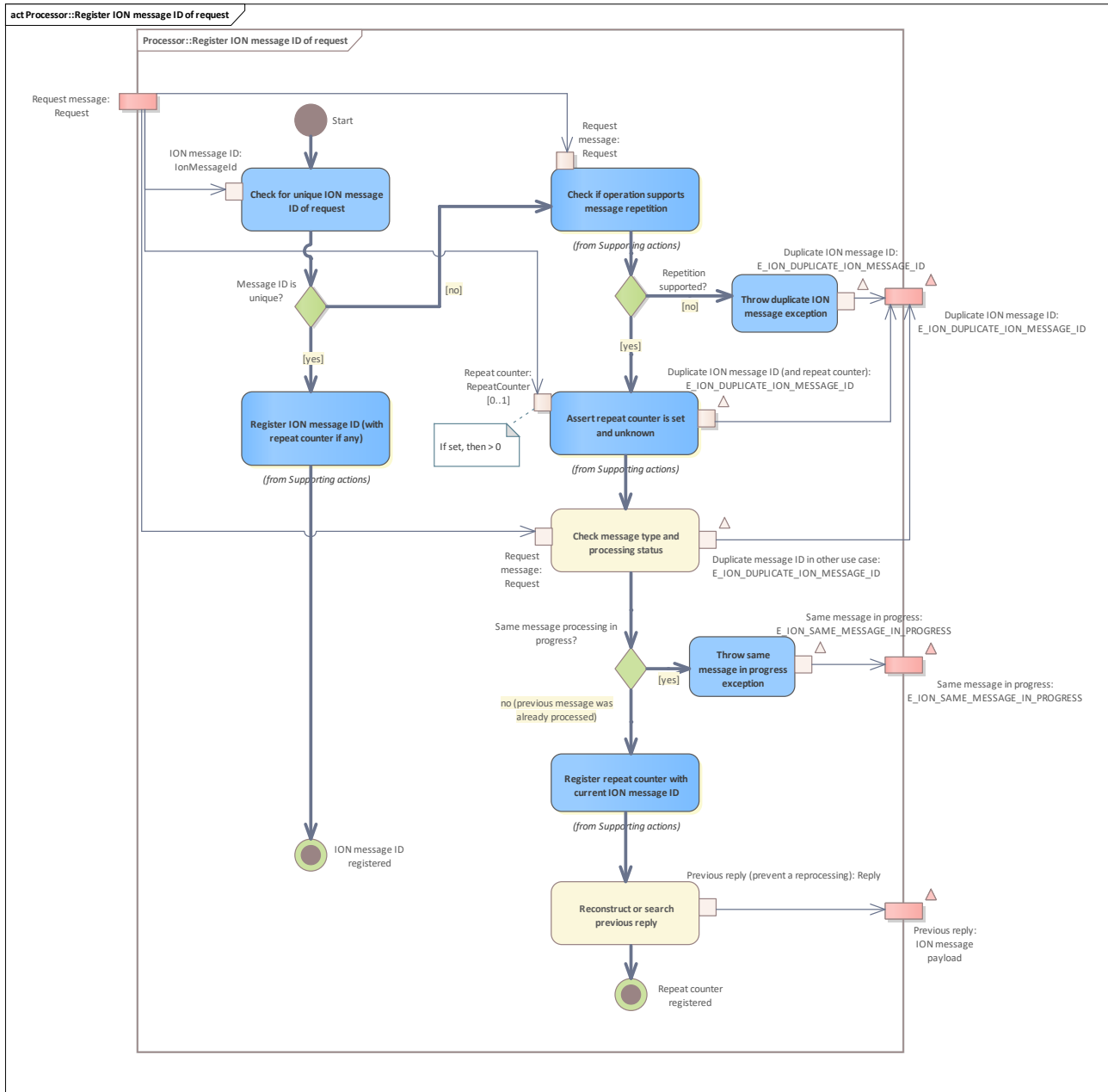
If the [RepeatCounter](#) is set to > 0, three different scenarios are possible:

1. The [RepeatCounter](#) is set, but the [IonMessageId](#) is unknown: register the [IonMessageId](#) and the [RepeatCounter](#) and proceed normally
2. The [RepeatCounter](#) is set, and the [IonMessageId](#) is known, the value of the [RepeatCounter](#) is not in the system: register the [RepeatCounter](#) with the existing [IonMessageId](#) and check the processing state: if already processed, prevent reprocessing and send the previous reply. If the processing of the same message is still in progress, stop the current process (and let the currently active process terminate gracefully and send the reply)

- The [RepeatCounter](#) is set, and the [IonMessageId](#) is known, the value of the [RepeatCounter](#) is already in the system: send an [E_ION_DUPLICATE_ION_MESSAGE_ID](#) to be embedded in a [BusinessException](#) and stop processing.

See also [Message Timeout and Retry Handling](#).

6.1.5.6.1 Processor::Register ION message ID of request



Activity diagram for [Register ION message ID of request](#).

Register ION message ID (with repeat counter if any)

Register the current [IonMessageId](#). If a repeat counter was set, store it also as a tuple with the current message ID.

Check message type and processing status

Check if a message with the same message ID has the same type. If not, then the same message ID was used in another use case context. In this case, an `E_ION_DUPLICATE_ION_MESSAGE_ID` is thrown since the repetition is not allowed. If the type matches, check if the same message is processed at the moment. If so, do not proceed but throw a fault and stop here (let the previous process sent the reply). This prevents sending more than one reply for the same message and avoids DoS attacks also.

Assert repeat counter is set and unknown

If the message ID was previously used, check if the repeat counter is set. The repeat counter belongs to the certain message ID and must be unique together with the message ID.

Reconstruct or search previous reply

In this step, it is clear that the same message has been processed previously. Thus a previous reply must exist or at least the data of a previous reply. This reply can be a regular response or a business exception.

Take the previous reply or reconstruct it using the existing data and return it as exception (to indicate that processing of the current message should be aborted) to the parent process.

Lookup in the datastore for the reply that has been sent before for the currently incoming request with the [IonMessageId](#).

The reply can be either a [BusinessException](#) or [Response](#), or an [exception payload](#) or [response payload](#), respectively.

The reply will include a current time stamp, and the repeat counter will be increased by one before it is sent again to the [Initiator](#).

Register repeat counter with current ION message ID

Register the repeat counter as a tuple with the current [IonMessageId](#) ([IonMessageId](#) and [RepeatCounter](#) - if set - together have to be unique).

Check for unique ION message ID of request

Check if the request message ID has not been used in any previous messages. Do not consider any repeat counter values in this check.

Check if operation supports message repetition

The WSDL documentation of the operation includes information regarding whether a specific operation must support message repetition.

The underlying concrete use case must correctly implement it.

In most cases, the repetition will not be needed in synchronous operations and will be needed in asynchronous operations.

Throw duplicate ION message exception

Throw same message in progress exception

6.1.5.7 Initiator::Register ION message ID of asynchronous reply

In the ION, each message has to be uniquely identified by a unique (composed) ID (see requirement [Unique identifier for each ION message](#)). If the system receives a request/response/exception message with an ID that already exists in the system, in most of the cases the message has to be refused. Then, the [IonMessageId](#) is not registered again and a (re-)processing of the message is rejected.

In the asynchronous response context, an [E ION DUPLICATE ION MESSAGE ID](#) is embedded in the [deliveryRejection](#).

The [IonMessageId](#) is also registered if a business exception is raised in the downstream business logic. This prevents having messages with the same ID, on one hand with a business acknowledgement and on the other hand with a business exception, as a response.

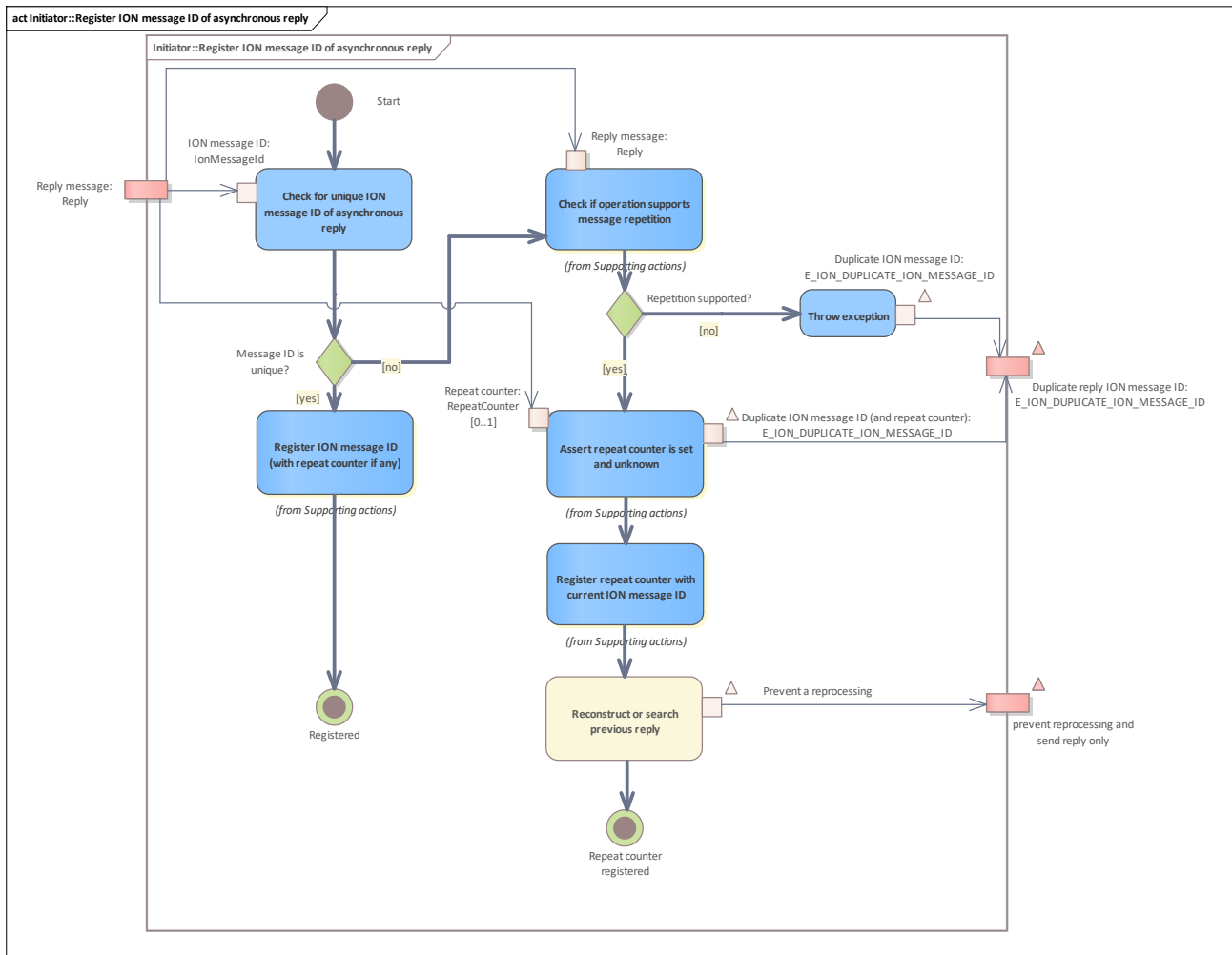
Thus, the message producer has to use a new ION message ID if a business exception has been received.

If the [RepeatCounter](#) is set to > 0, three different scenarios are possible:

1. The [RepeatCounter](#) is set, but the [IonMessageId](#) is unknown: register the [IonMessageId](#) and the [RepeatCounter](#) and proceed normally
2. The [RepeatCounter](#) is set, and the [IonMessageId](#) is known, the value of the [RepeatCounter](#) is not in the system: register the [RepeatCounter](#) with the existing [IonMessageId](#) and check the processing state: if already processed, prevent reprocessing and send the previous reply. If the processing of the same message is still in progress, stop the current process (and let the currently active process terminate gracefully and send the reply)
3. The [RepeatCounter](#) is set, and the [IonMessageId](#) is known, the value of the [RepeatCounter](#) is already in the system: send an [E ION DUPLICATE ION MESSAGE ID](#) to be embedded in a [deliveryRejection](#) and stop processing.

See also [Message Timeout and Retry Handling](#).

6.1.5.7.1 Initiator::Register ION message ID of asynchronous reply



Activity diagram for [Register ION message ID of asynchronous reply](#).

Check for unique ION message ID of asynchronous reply

Check if the message ID of the asynchronous reply has not been used in any previous messages. Do not consider any repeat counter in this check.

Register ION message ID (with repeat counter if any)

Register the current [IonMessageId](#). If a repeat counter was set, store it also as a tuple with the current message ID.

Register repeat counter with current ION message ID

Register the repeat counter as a tuple with the current [IonMessageId](#) ([IonMessageId](#) and [RepeatCounter](#) - if set - together have to be unique).

Assert repeat counter is set and unknown

If the message ID was previously used, check if the repeat counter is set. The repeat counter belongs to the certain message ID and must be unique together with the message ID.

Reconstruct or search previous reply

In this step, it is clear that the same message has been processed previously. Thus, a previous reply must exist or at least the data of a previous reply. This reply can be a regular [deliveryAcknowledgement](#) or a [deliveryRejection](#).

Take the previous reply or reconstruct it using the existing data and return it as exception (to indicate that processing of the current message should be aborted) to the parent process.

Check if operation supports message repetition

The WSDL documentation of the operation includes information regarding whether a specific operation must support message repetition.

The underlying concrete use case must correctly implement it.

In most cases, the repetition will not be needed in synchronous operations and will be needed in asynchronous operations.

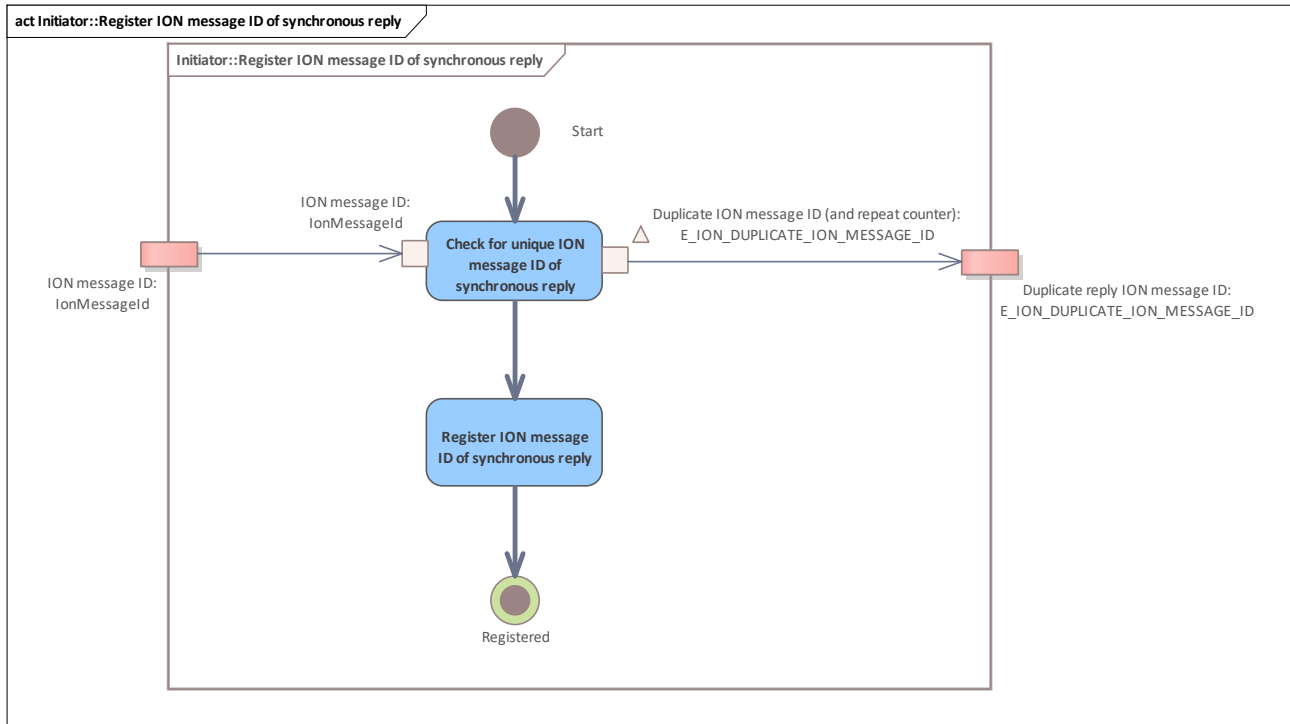
Throw exception

6.1.5.8 Initiator::Register ION message ID of synchronous reply

In the ION, each message has to be uniquely identified by a unique (composed) ID (see requirement [Unique identifier for each ION message](#)). If the system receives a synchronous response or exception message with an ID that already exists in the system, the message has to be refused. Then, the [IonMessageId](#) is not registered again and the reply (response or exception) will not be processed. Any possible [RepeatCounter](#) values are ignored. The case must be clarified manually.

See also [Message Timeout and Retry Handling](#).

6.1.5.8.1 Initiator::Register ION message ID of synchronous reply



Activity diagram for [Register ION message ID of synchronous reply](#).

Register ION message ID of synchronous reply

Ignore any repeat counter for synchronous replies. Store only the [IonMessageId](#).

Check for unique ION message ID of synchronous reply

Check if the message ID of the synchronous reply has not been used in any previous messages. Do not consider any repeat counter in this check. In the synchronous reply context, the [repeat counter](#) does not make any sense. If the message ID is not unique, throw a [E ION DUPLICATE ION MESSAGE ID](#) and treat it internally or manually.

6.1.6 Supporting actions

This package contains actions that are needed in more than one use case for ION messaging.

6.1.6.1 Example actions

This package contains example actions that are used to build example activities, which are embedded in the ION context and are placeholders for real activities in the business model.

6.1.6.1.1 First check (potential exception)

Example for a check which is done with the [Request business data](#). This example check may raise a [BusinessException](#).

6.1.6.1.2 Second check (potential event)

Example for a check which is done with the [Request business data](#). This example check may raise an [Event](#).

6.1.6.1.3 Third check (potential event)

Example for a check which is done with the [Request business data](#). This example check may raise a further [Event](#).

6.1.6.1.4 Fourth check (asynchronous)

Action that is to show an asynchronous test step. For example, a system user activity may be required, a request to an external system or similar.

If such actions occur in an activity, the entire use case becomes asynchronous.

6.1.6.2 Exceptions

This package contains non-specified exceptions which may occur if technical problems concerning security or message syntax are detected.

6.1.6.2.1 SOAP fault occurred

Handle a [SOAP fault](#). Normally, the underlying process will be aborted.

6.1.6.2.2 Wrong receiver

Potential runtime exception if the intended receiver is not me. In the synchronous context, a business exception is not possible.

6.1.6.2.3 Duplicate response message ID

Potential runtime exception if the response message ID is already in the system. In the synchronous context, a business exception is not possible.

6.1.6.2.4 Response/request mismatch

Potential runtime exception if the correlation in the response's [OriginalRequestCommunicationInformation](#) does not match the current request message ID ([IonMessageId](#)). In the synchronous context, a business exception is not possible.

6.1.6.2.5 Invalid security

Exception if the message cannot be decrypted or that the message's signature cannot be verified.

The most commonly used error message in the different WSS frameworks is "Invalid security header".

In most of these cases, a wrong key reference is sent, the normalisation of the message is wrong or the message does not meet the requirements of the etiCORE WS policy (see [Appendix: etiCORE Standard-Policy](#)).

The underlying framework has to respect the requirement [Security relevant errors](#). If the framework does not fulfil it, a separate exception handler must be employed.

6.1.6.2.6 Invalid payload

Exception if an automatic schema validation error occurs.

Two scenarios may occur:

- If the automatic schema validation of the incoming message fails, the software implementation of the sender has to be fixed. The called operation terminates with this exception and no processing takes place
- If the automatic schema validation of the outgoing message fails, your software implementation or data storage has to be fixed. The process stops with this exception and prevents the case from above

For incoming messages, the underlying framework has to respect the requirement [Security relevant errors](#) to avoid DoS attacks. If the framework does not fulfil it, a separate exception handler must be employed.

6.1.6.3 Assign message correlation ID to original request

Check the correlation between the original request which caused the current response (also the current business exception, if the incoming answer is an exception) and this current incoming response by comparing the field "originalRequestCorrelationId" of the [OriginalRequestCommunicationInformation](#) in the [BusinessAcknowledgement](#) or [BusinessException](#) with message IDs of the own request history.

Additionally, check if the current sender, sender role and sender service match the original receiver, receiver role and receiver service of the original request.

If no original request matches all checks from above, throw an [E IONC ORIGINAL REQUEST NOT FOUND](#) exception.

If a previous response exists, which is already correlated with the original request, throw an [E IONC DUPLICATE RESPONSE](#) exception.

6.1.6.4 Automatic Schema validation

Do the automatic XML schema validation with the embedded payload.
Normally performed by the underlying web service framework.

6.1.6.5 Build message header

Build the message header which is used in the SOAP message or [SOAP Envelope](#). This header is defined by [IonRoutingHeader](#) and will be embedded in the [SOAP Header](#).

The operation name needed in the [IonRoutingHeader](#) is to be derived from the request payload. Due to WS-I compatibility, this can be done very easily.

Example: Payload is called *notifyEntitlementIssued* -> operation name will be the same: *notifyEntitlementIssued*.

Furthermore, the interface version in the new [IonRoutingHeader](#) must be set to the one of the system that currently builds the message header.

In the asynchronous reply context, the [OptionalRoutingInfo](#) in the new [IonRoutingHeader](#) must be the same as in the original request of the [Initiator](#) (copy from request).

6.1.6.6 Build response communication information

Complete the "existing" payload's communication information.

To build the [CommunicationInformation](#) object of the response:

- sender ID from request [IonMessageId](#) -> response receiver ID
- sender role from request [IonMessageId](#) -> response receiver role
- sender service from request [IonMessageId](#) -> response receiver service

- new [IonMessageId](#) for the response
- new message timestamp
- no [RepeatCounter](#)!
- request [ProcessInstanceId](#) -> response process instance ID

Take the following fields from [CommunicationInformation](#) of the original request and copy them to the response payload to build the [OriginalRequestCommunicationInformation](#):

- [IonMessageId](#)
- Timestamp
- [RepeatCounter](#) (if present)

6.1.6.7 Build SOAP message

Build the SOAP message employing a [SOAP Envelope](#) with consideration of the binding rules in the corresponding WSDL. These rules determine if an [IonRoutingHeader](#) has to be embedded in the [SOAP Header](#).

Furthermore, the embedded webservice security policy in the WSDL is used to sign and encrypt the necessary parts of the message.

6.1.6.8 Check if the response corresponds to the request

In synchronous context, assert the equality of the [OriginalRequestCommunicationInformation](#)'s copied original request [IonMessageId](#) and timestamp in the response with the corresponding fields in the [CommunicationInformation](#) of the current request.

6.1.6.9 Check implementation for message processing

Interceptor that performs a lookup for the operation that implements the processing of the incoming message. Due to the etiCORE release management, certain functionality may not be implemented, since it belongs to a higher minor version that is already implemented by the initiator but not by the processor.

The interceptor may work either with the header data or the payload. This depends on the position of the interceptor in the processing chain and is not determined here.

This action must be done before the schema validation runs.

6.1.6.10 Check message

Check if the SOAP message is well-formed and complies with the standard SOAP schema and the binding defined in the WSDL:

- SOAP message in general
- SOAP header composition
- WS-policy

Normally performed by the underlying framework.

6.1.6.11 Check message signature

Check the compliance to the WSS policy described in the chapter [Message-based encryption \(application layer\)](#) and [Appendix: etiCORE Standard-Policy](#).

Check the signature with the right certified public key from the directory service of the PKI. Identify the certificate matching all of the following parameters

- the organisation ID of the sender (with certificate subject)
- the role of the sender (with certificate subject)
- the purpose ("sig"), see [Basic Requirements](#)
- the serial number of the key located in the security header

Furthermore, consider the rules concerning the [Validity of Certificates](#).

Check that the signature was performed by the sender and its role is located in the SOAP Header.

Partially performed by the underlying web service framework's security provider.

Note: due to the etiCORE [Policy](#), only the signature key's serial number is contained in the [Security header](#) of the [SOAP header](#) in clear text.

Note: consider the [Transition Phase for Certificate Renewal](#) and [Security Levels](#).

6.1.6.12 Check receiver equals "me"

The receiver of the message must be my organisation ID and my role (see [PartnerRoleEnum](#)) and my service.

This check for organisation and role will never fail as long as the WSS is switched on and only becomes relevant without WSS.

The check for the service name may fail even if WSS is switched on since the service name is not part of the WSS parameters. A service name mismatch may only occur if a partner configured a wrong URL for its service.

6.1.6.13 Check sender is known

Check if the sender of the message exists in your own system's master data.

Note: each participating system is obliged to update its master data regularly by requesting the [Registrar](#) of the [Scheme Manager](#) for the organisation and role list of all participants.

6.1.6.14 Check sender has the right role and service

Check if the sender has the right role and the right service to perform this use case and call the current operation.

This is important in the asynchronous scenario since the processor will try to reply to the sender role and service. If the sender service does not provide appropriate response / exception operations, replies can not be routed.

The list of services allowed to call an operation is part of the documentation of the operation in the WSDL.

6.1.6.15 Check TLS client certificate

The sender has to provide its TLS certificate.

All certificates for ION message exchange are issued by one defined authority and have a certain validity.

To verify the client certificate, an OCSP call has to be made to the trust centre or a current certificate revocation list has to be evaluated. Furthermore, the complete certificate chain has to be evaluated. Each certificate of the chain must be valid.

Normally, this is performed in the upstream web server of the receiving system.

See also [Transport encryption via TLS](#) and [Validity of Certificates](#).

Additionally, this check has to ensure that the right security level is used, see also [Security Levels](#).

Note: If the receiving party is a participant system or a JSB, the TLS client certificate should be always the one of the CRE. If the receiving party is the CRE, it has to verify the TLS client certificate of the participant system or the JSB.

6.1.6.16 Convert to delivery rejection

Embed the incoming [EventIdentifier](#) into a [deliveryRejection](#) with the specified format in [Asynchronous Reply Overview](#).

6.1.6.17 Convert to exception payload

Take an incoming [EventIdentifier](#), create the matching exception element including the correlation information of the original request without certain fields ([IonMessageId](#), etc.) of the [CommunicationInformation](#) which have to be filled later when the exception is sent.

6.1.6.18 Convert to SOAP fault

Embed the incoming [EventIdentifier](#) into a [SOAP fault](#) with the specified format in [Asynchronous Reply Overview](#).

6.1.6.19 Create message ID

Create a new [IonMessageId](#) which is embedded in the communication information of the payload.

6.1.6.20 Create request

Create the request as an XSD top level element containing the [Request business data](#), together with the [CommunicationInformation](#) as far as possible. Do not consider the creation of the [IonMessageId](#) or [ProcessInstanceId](#). This is done later during the ION message compilation. The own organisation ID, as well as the own role, is also added later in the ION message compilation, see [Compile and send ION request](#). The receiver role (and service) has to be set only if the use case does not determine it.

6.1.6.21 Create response

Template action to create a [Response payload](#) object.

Compile [Response business data](#) (if any beyond the [BusinessAcknowledgement](#)).

Build a top-level element for the response payload containing

- "empty" [OriginalRequestCommunicationInformation](#) and [CommunicationInformation](#) objects¹
- the [WarningList](#) with the processing events - if any
- the newly compiled [Response business data](#) - if any

¹To keep the same steps of filling the [OriginalRequestCommunicationInformation](#) and [CommunicationInformation](#) objects in the response payload away from the business logic, this is done later in [Processor::Compile and send an asynchronous ION response](#).

Also do not consider the (new) response ION message ID and your own organisation ID and role. These are also done there.

6.1.6.22 Decrypt Message

Check the compliance to the WSS policy described in the chapter [Message-based encryption \(application layer\)](#) and [Appendix: etiCORE Standard-Policy](#).

Decrypt the incoming SOAP message using your own private key for

- the organisation
- the role
- the purpose (here: decryption)

Only the SOAP body is encrypted. The encryption information is located in the security header embedded in the SOAP header.

Normally performed by the underlying web service framework's security provider.

Note: consider the [Transition Phase for Certificate Renewal](#) and [Security Levels](#).

6.1.6.23 Determine the receiver

E.g. in notification scenarios, the receiver organisation can be extracted from the contained binary transaction attestation. The receiver role comes from the use case context (e.g. a notification about an entitlement blocking is always initially sent to the PO) based on the information of the intended called operation that was called.

Note: One organisation may have more than one role.

6.1.6.24 Determine reply type and convert it to ION message

Action that rebuilds an ION message for an existing [message payload](#) of a reply. Depending on the message type, either an [asynchronous ION response message](#) or an [asynchronous ION exception message](#) will be built and passed back in order to be directly sent to the sender.

6.1.6.25 Do any business exception handling in the initiator system

After an [asynchronous ION exception message](#) was received and acknowledged, there may be - depending on the kind of exception - necessary post processing, potentially asynchronously, too.

6.1.6.26 Do any post-processing in the initiator system

Do post-processing depending on the underlying use case.

If the [Response payload](#) contains [Response business data](#), this business data must be processed here.

Any other post-processing is also possible here. This might be something out of scope like sub-ledger steps or something specified in etiCORE like further ION calls or calling an included use case.

6.1.6.27 Encrypt message

Encrypt the message with the receiver's public key. This public key is certified and can be obtained from the PKI by its directory service.

Note: please consider the rules concerning the [Validity of Certificates](#).

Due to the web service security [Policy](#), the entire [SOAP Body](#) has to be encrypted. The [SOAP Header](#) remains in clear text for routing purposes. Furthermore, the signature of the [SOAP Body](#)

also has to be encrypted. The encrypted signature data of the signed [SOAP Body](#) is stored in the [Security header](#) in the [SOAP Header](#) and is not part of the [SOAP Body](#).

6.1.6.28 Extract response payload

Normally performed by the underlying web service framework. The payload is the etiCORE defined embedded top-level element in the [SOAP Body](#) always as inherited [BusinessAcknowledgement](#) and, optionally, implements [Response business data](#) (here: abstract placeholder for all implementing business data response classes).

6.1.6.29 Extract previous delivery state

Extract the previous reply concerning the delivery state of the message. Return either the contained [deliveryAcknowledgement](#) or [deliveryRejection](#).

6.1.6.30 Finalise response payload

Complete the "incomplete" response payload by adding the [OriginalRequestCommunicationInformation](#) object and the newly built [CommunicationInformation](#) object.

6.1.6.31 Finalise exception payload

Complete the "incomplete" [exception payload](#) by adding the [OriginalRequestCommunicationInformation](#) object and the newly built [CommunicationInformation](#) object.

6.1.6.32 Handle delivery rejection

Action that handles a potential [deliveryRejection](#). This reply has to be evaluated and then, the right actions must be taken depending on the use case.

6.1.6.33 Pass exception payload to business logic

Allow the exception - represented by the exception payload - to be returned to the higher-level business logic after the generic checks failed.

6.1.6.34 Pass response payload to business logic

Allow the response - represented by the response payload - to be returned to the higher-level business logic after the generic checks have been performed.

6.1.6.35 Process and collect business data

Abstract action that stands for the concrete business logic which collects the necessary data in one or more steps.

If no business data is needed (e.g. for pure get-scenarios), this step will only indicate which data is needed from a third-party system (getXXX).

6.1.6.36 Open Connection

Opens a TLS connection (to the CRE) presenting the [ION TLS certificate](#) . In case the CRE URL uses the HTTP scheme, open TCP connection or re-use an existing one.

Opens a TLS connection to the CRE presenting the [ION TLS certificate](#) found in the [ION Keystore](#) (see also [Keys and certificates : ION Key- and Truststore](#)) for the configured organisation ID. Checks the validity of the retrieved certificate against the [ION Truststore](#) (incorporating CRLs or OCSP). Asserts that the organisation ID given in the "o" part of the certificate subject matches 5602 (organisation ID of the CRE) and that there is no "ou" part (i.e. it is an TLS certificate).

6.1.6.37 Open Connection

See [Open Connection](#)

Initiator::Register ION message ID of synchronous reply

See [Initiator::Register ION message ID of synchronous reply](#).

6.1.6.38 Send delivery acknowledgement

Send a [DeliveryAcknowledgement](#) to

- the [Processor](#), if a [Request](#) has been received
- the [Initiator](#), if a [Reply](#) has been received

with a `<deliveredTo>RECIPIENT</deliveredTo>`.

Always done in an asynchronous context if the decryption and signature verification, the automatic schema validation and some basic checks were successful.

6.1.6.39 Sign message

Sign the message parts as defined in the [Webservice Security Policy](#) with the private key for signature purposes (the corresponding public key is registered and certified in the PKI and can be accessed via the directory service of the PKI).

According to the web service security specification, the entire body of the SOAP message which contains the payload is signed.

Normally performed by the underlying WSS framework.

6.1.6.40 Update delivery status in initiator system

Register [deliveryAcknowledgement](#) as preliminary confirmation, so that the request can be processed in the [Processor](#)'s business logic. The next step is to wait the SLA timespan for the final business response (see [\[5\] SLAs for Message Transfer](#)).

6.1.6.41 Update status in the initiator system

A business response or exception is received and status is updated accordingly in the initiator's system.

The generic message correlation steps are done before in the use cases [Receive and validate asynchronous ION response](#), [Receive and validate asynchronous business exception](#), [Receive and validate synchronous ION response](#) or [Receive and validate synchronous ION business exception](#).

This step should update the status of a certain entity depending on the underlying use case (e.g. set the state of an entitlement to "hotlisted").

In the case of a business exception: depending on exception type, a subsequent handling may be required after updating the status to enable regular processing in further actions. **It is assumed that the problem defined in the exception is solved before the action completes. Only then, the follow-up steps can continue.**

In the case of warnings contained in the reply, a further warning handling could be necessary. If the [response payload](#) contains [Response business data](#), this business data must be processed in further steps.

6.1.6.42 Update status in the processor system

Update the status in the processor's system concerning the processed request and the response sent to the initiator. The final status update is the same for synchronous or asynchronous contexts.

In the case of an incoming [deliveryRejection](#), consider a manual scenario.

6.1.6.43 Update error status in processor system

Update the error status registering the occurred error and the reference to the sent exception.

In the case of an incoming [deliveryRejection](#), consider a manual scenario.

6.2 CRE Use Cases

This chapter contains the use cases of the central routing engine (CRE).

6.2.1 Overview of the central routing engine use cases

This chapter provides an overview of the use cases of the Central Routing Engine and shows their relationships and interdependencies.

(Includes)	
Linked Use Cases	
(Realises)	
Base Activity	
Inputs	
Outputs	
Error Cases	
Activity Diagram	

6.2.1.2 Diagram: Overview of the central routing engine use cases

This diagram shows the main use cases of the central routing engine where back-office systems are directly involved.

The following use cases are based on a real service with the CRE serving as the [Processor](#) for notifying the availability of services:

- [Handle request to set service as available for a participant](#)
- [Handle request to set service as unavailable for a participant](#)

The following use case is based on a real service that is used by the CRE as [Initiator](#):

- [Notify about discarded messages](#)

The following use cases serve as a template to show the common workflow when an ION message is processed:

- [Process synchronous message](#)
- [Process asynchronous message](#) with its extensions [Store asynchronous ION message](#) and [Deliver queued ION messages](#) when the service is available.

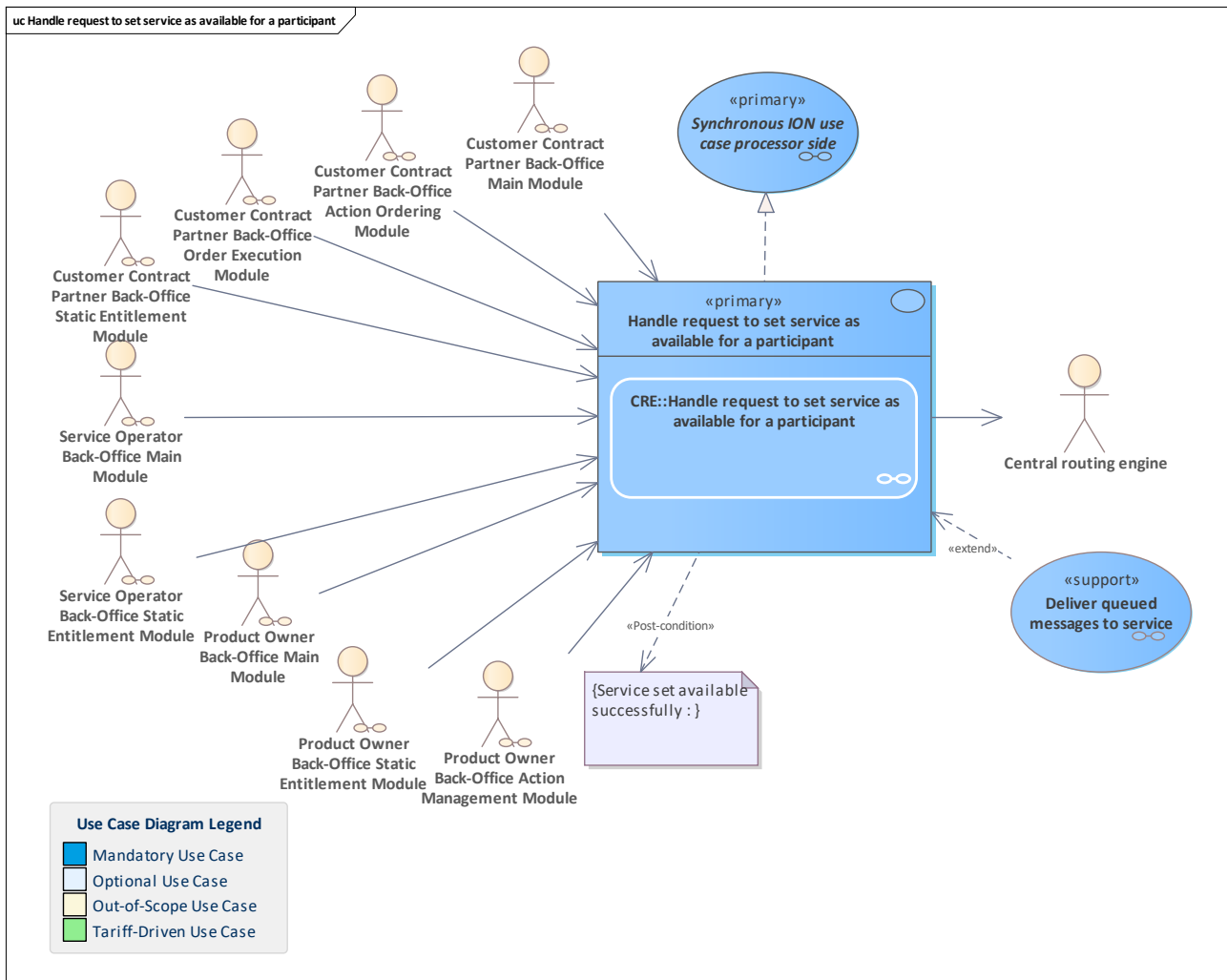
The use case [Deliver queued messages in scheduler](#) shows when messages are redelivered to the intended receiver within the CRE's scheduler.

6.2.2 Service availability notification

Contains the use cases and activities for handling service (non-)availability notifications from the participating back-office systems.

Each organisational unit (organisation + role) can define its URLs where its systems are available. This can be done for the standard service of the role as well as for the extensions that are covered as own services.

6.2.2.1 Handle request to set service as available for a participant



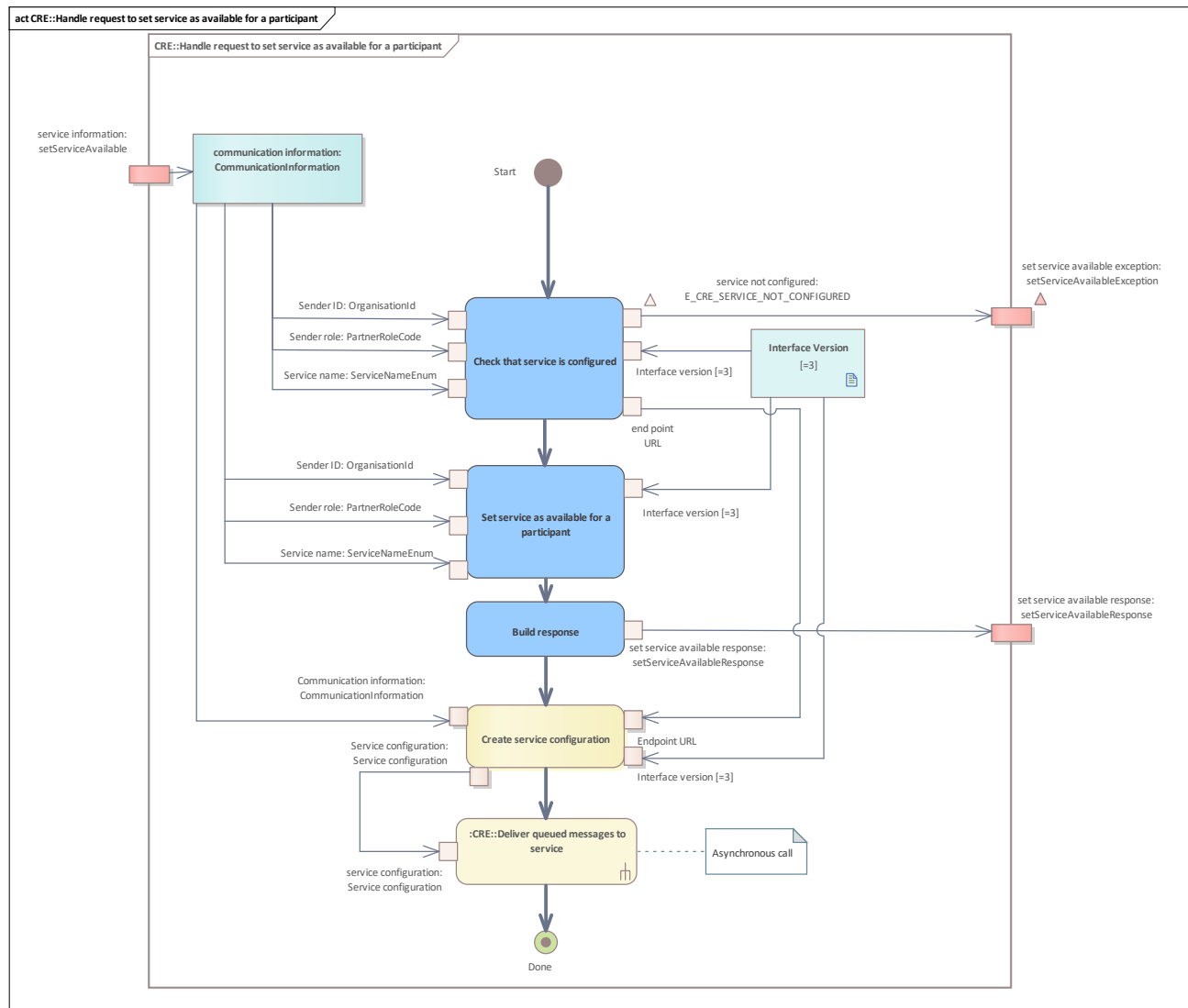
Use Case	Handle request to set service as available for a participant
Description	<p>Use case the on CRE side for a participant to set its service to available for receiving asynchronous messages. If the participant provides more than one service, it has to call the use case operation for each of its provided service.</p> <p>With this use case, the participant indicates to the CRE that this service is available at its configured URL and can be used by all other participants.</p> <p>If any asynchronous messages have been stored temporarily during the offline period for this participant and service, this use case will trigger the use case Deliver queued messages to service. Announcing own services being available is necessary in the asynchronous context to indicate that the system is ready to receive messages on its configured URL (for configuration, see [4] Documentation Registrar System for Service Configuration).</p> <p>Note: Synchronous messages cannot be queued temporarily. Therefore, in this context, notifying the availability of a service that works exclusively with synchronous messages does not make sense.</p> <p>Note: this use case works with WSS like all further use cases.</p>
Initiating Actor	Initiator Processor

	Customer Contract Partner Back-Office Main Module Customer Contract Partner Back-Office Action Ordering Module Customer Contract Partner Back-Office Order Execution Module Product Owner Back-Office Main Module Product Owner Back-Office Action Management Module Service Operator Back-Office Main Module Product Owner Back-Office Static Entitlement Module Service Operator Back-Office Static Entitlement Module Customer Contract Partner Back-Office Static Entitlement Module
Reacting Actor	Central routing engine
Preconditions	
Postconditions	Service set available successfully
Linked Use Cases (Extended By)	Deliver queued messages to service
Linked Use Cases (Includes)	
Linked Use Cases (Realises)	Synchronous ION use case processor side
Base Activity	
Inputs	service information : setServiceAvailable
Outputs	set service available response : setServiceAvailableResponse
Error Cases	E_CRE_SERVICE_NOT_CONFIGURED set service available exception : setServiceAvailableException
Activity Diagram	CRE::Handle request to set service as available for a participant

6.2.2.1.1 CRE::Handle request to set service as available for a participant

Activity for [Handle request to set service available for participant](#).

6.2.2.1.1.1 CRE::Handle request to set service as available for a participant



Activity diagram for [Set service available for a participant](#)

Check that service is configured

The new state can only be switched to available if the specified service exists and is configured (URL exists) for the calling [Initiator](#).

Set service as available for a participant

Switch the state for the service to available.

From now on - assuming the service is really available - asynchronous messages can and will be routed to the service URL. These messages will no longer be queued in the CRE but sent directly to the configured service.

Build response

Build the response to indicate that the service has been set to *available* using [setServiceAvailableResponse](#).

CRE::Deliver queued messages to service

See [CRE::Deliver queued messages to service](#).

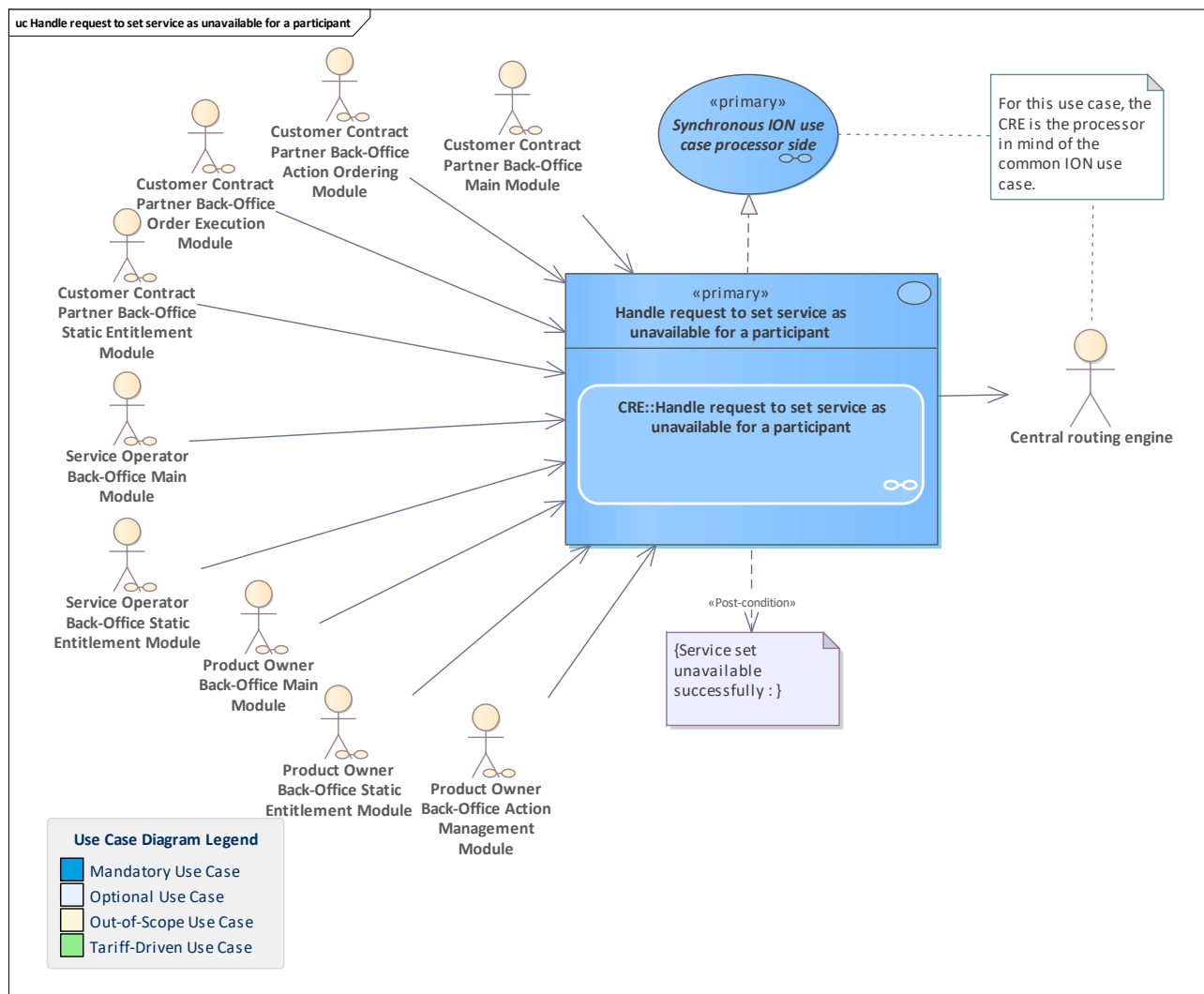
Create service configuration

Create or assemble service configuration consisting of

- Interface version (only major version)
- Receiver organisation ID
- Receiver role
- Receiver service
- The related endpoint URL

This configuration can be used to filter and send messages from the re-delivery queue.

6.2.2.2 Handle request to set service as unavailable for a participant

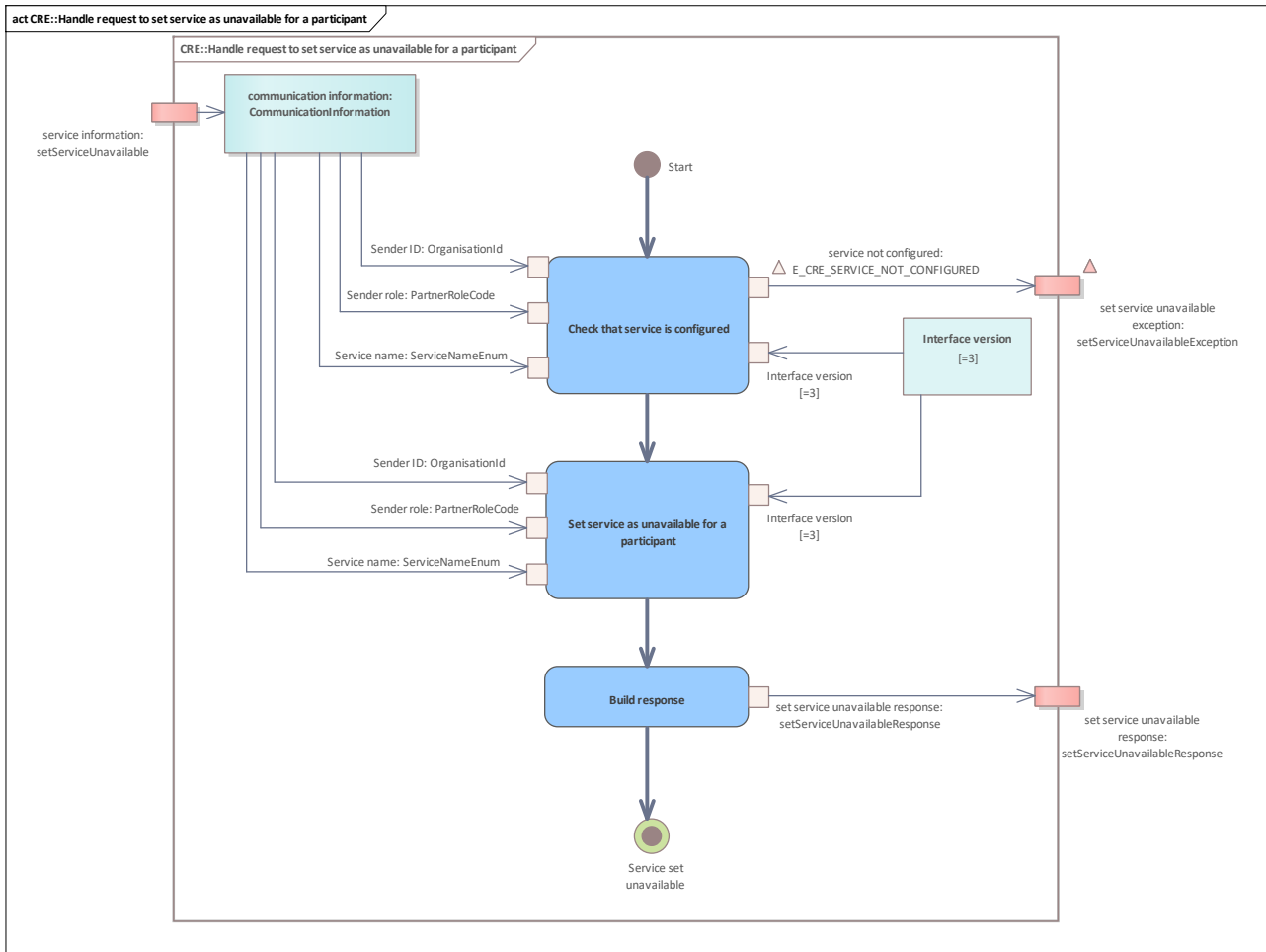


Description	<p>Use case on CRE side for a participant to announce its service as unavailable for receiving asynchronous messages. If the participant provides more than one service, it has to call the use case operation for each of its provided services.</p> <p>After the state transition of this service, the participant indicates that this service is no longer available at its configured URL to all other participants . It is very important to indicate the unavailability of the system or service. It prevents the CRE from sending messages frequently and produces error messages, normally E IONC RECEIVER NOT REACHABLE.</p> <p>If asynchronous messages are sent to this participant's service, they will be stored temporarily by the CRE until the service is announced to be available again by Handle request to set service available for participant.</p> <p>Note: Synchronous messages cannot be queued temporarily. Therefore, in this context, notifying the unavailability of a service that works exclusively with synchronous messages does not make sense and will not stop the routing of messages.</p> <p>Note: this use case works with WSS like all further use cases.</p>
Initiating Actor	Processor Initiator Customer Contract Partner Back-Office Main Module Customer Contract Partner Back-Office Action Ordering Module Customer Contract Partner Back-Office Order Execution Module Customer Contract Partner Back-Office Static Entitlement Module Service Operator Back-Office Main Module Service Operator Back-Office Static Entitlement Module Product Owner Back-Office Main Module Product Owner Back-Office Static Entitlement Module Product Owner Back-Office Action Management Module
Reacting Actor	Central routing engine
Preconditions	
Postconditions	Service set unavailable successfully
Linked Use Cases (Extended By)	
Linked Use Cases (Includes)	
Linked Use Cases (Realises)	Synchronous ION use case processor side
Base Activity	
Inputs	service information : setServiceUnavailable
Outputs	set service unavailable response : setServiceUnavailableResponse
Error Cases	E CRE SERVICE NOT CONFIGURED set service unavailable exception : setServiceUnavailableException
Activity Diagram	CRE::Handle request to set service as unavailable for a participant

6.2.2.2.1 CRE::Handle request to set service as unavailable for a participant

Activity for [Handle request to set service unavailable for a participant](#).

6.2.2.2.1.1 CRE::Handle request to set service as unavailable for a participant



Activity diagram for [Handle request to set service unavailable for a participant](#).

Check that service is configured

The new state can only be switched to unavailable if the specified service exists and is configured for the calling [Initiator](#).

Set service as unavailable for a participant

Switch the state for the service to unavailable.
From now on asynchronous messages will no longer be routed to the service URL. These messages will now be queued in the CRE.

Build response

Build the response to indicate that the service has been set to *unavailable* using [setServiceUnavailableResponse](#).

6.2.3 Conceptual routing

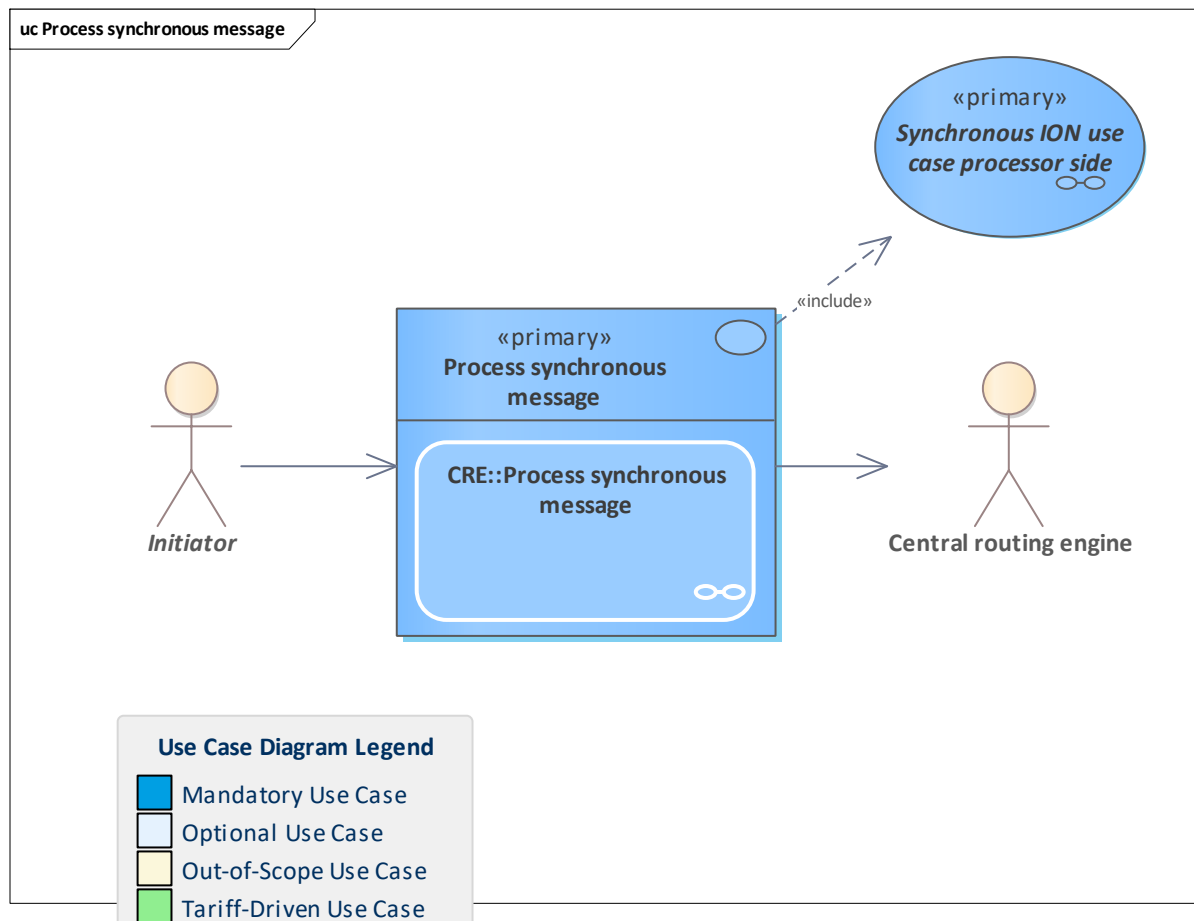
Contains use cases which describe the routing of messages for synchronous and asynchronous context.

For explanatory purposes, the use cases and diagrams show the exchange of fictional messages (serving as placeholders for all real messages).

To implement the ION specification for a back-office system, the content of this chapter is not mandatory. But it may help to understand the message handling in the ION and the CRE's routing mechanisms.

However, to implement the CRE, this chapter is important.

6.2.3.1 Process synchronous message



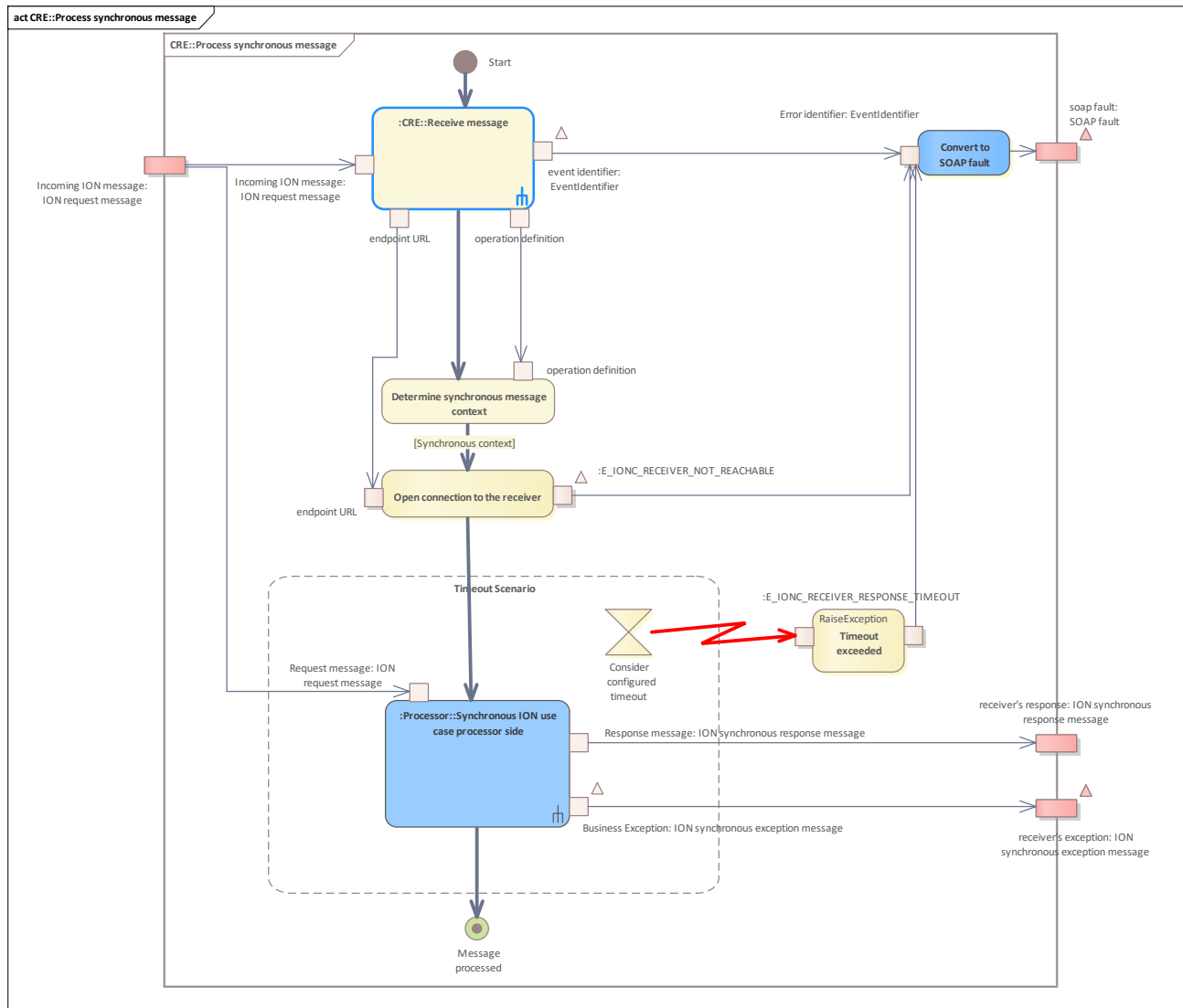
Use Case	Process synchronous message
Description	<p>Process a message in a synchronous context.</p> <p>The CRE forwards the message directly from the Initiator to the Processor.</p> <p>Message caching is not possible. If the processor is not available, an exception is sent to the initiator.</p> <p>The CRE keeps the communication channel open until the message exchange is completed or the configured timeout occurred.</p> <p>The response of the processor is returned directly via the open communication channel.</p>
Initiating Actor	Initiator
Reacting Actor	Central routing engine
Preconditions	
Postconditions	

Linked Use Cases (Extended By)	
Linked Use Cases (Includes)	Synchronous ION use case processor side
Linked Use Cases (Realises)	
Base Activity	
Inputs	Incoming ION message : ION request message receiver's response : ION synchronous response message
Outputs	
Error Cases	E ION RECEIVER ROLE MISMATCH E ION RECEIVER SERVICE MISMATCH E ION RECEIVER ORGANISATION MISMATCH E ION UNKNOWN SENDER E ION WRONG SENDER ROLE E ION WRONG SENDER SERVICE E ION SAME MESSAGE IN PROGRESS E ION DUPLICATE ION MESSAGE ID E ION DUPLICATE ION MESSAGE ID E ION DUPLICATE ION MESSAGE ID receiver's exception : ION synchronous exception message soap fault : SOAP fault
Activity Diagram	CRE::Process synchronous message

6.2.3.1.1 CRE::Process synchronous message

Activity for [Process message in a synchronous context](#).

6.2.3.1.1.1 CRE::Process synchronous message



Activity diagram for [Process message in a synchronous context](#).

Open connection to the receiver

Opens the connection and do the TLS handshake with the receiver including the check of the receiver's TLS certificate.

If a communication problem occurs, an [E_IONC_RECEIVER_NOT_REACHABLE](#) occurs.

Determine synchronous message context

Find out, if the message is sent in a synchronous or asynchronous context. This is configured inside the CRE for each operation to be called for the message. Here we assume that the synchronous context was configured.

Consider configured timeout

For each operation - independent from synchronous or asynchronous context - a timeout is configured inside the CRE by which the accessed system has to respond.

The default value can be found in [Appendix: CRE Timeout Configuration Table](#).

Timeout exceeded

Raises an exception that the configured timeout has been exceeded. This results in an [E IONC RECEIVER RESPONSE TIMEOUT](#) towards the sender.
For asynchronous contexts, the message is **not queued** in the CRE.

Processor::Synchronous ION use case processor side

See [Processor::Synchronous ION use case processor side](#).

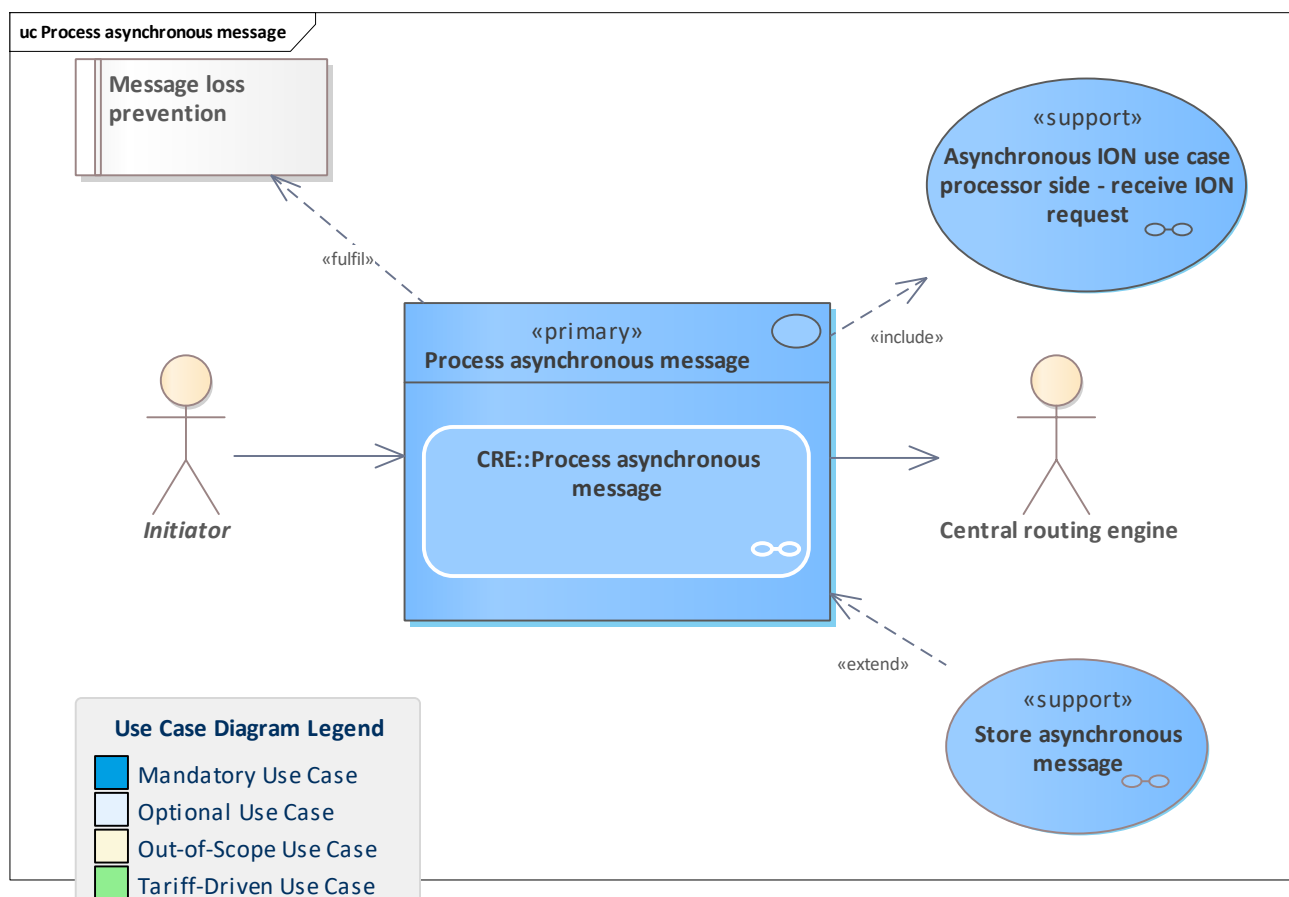
CRE::Receive message

See [CRE::Receive message](#).

Convert to SOAP fault

Embed the incoming [EventIdentifier](#) into a [SOAP fault](#) with the specified format in [Asynchronous Reply Overview](#).

6.2.3.2 Process asynchronous message



Use Case	Process asynchronous message
Description	Process message in an asynchronous context. The internal configuration in the CRE of the message parameters indicates that a message is sent in an asynchronous context. For the CRE, there is no difference if the message is sent from the Initiator to the

	<p>Processor or the asynchronous response is sent from the processor back to the initiator. From a technical viewpoint, there is no difference for the routing. The CRE is stateless and does not correlate the first message with the second message.</p> <p>The only important reason to know if the context for the message is asynchronous is to perform message caching if the target system is not available.</p> <p>In this case, Store asynchronous ION message is performed.</p>
Initiating Actor	Initiator
Reacting Actor	Central routing engine
Preconditions	
Postconditions	
Linked Use Cases (Extended By)	Store asynchronous message
Linked Use Cases (Includes)	Asynchronous ION use case processor side - receive ION request
Linked Use Cases (Realises)	
Base Activity	
Inputs	delivery acknowledgement : deliveryAcknowledgement incoming ION message : ION message with WSS
Outputs	
Error Cases	delivery rejection : deliveryRejection
Activity Diagram	CRE::Process asynchronous message

6.2.3.2.1 CRE::Process asynchronous message

Activity for [Process asynchronous message](#).

```
act CRE::Process asynchronous message
```



Opens the connection and do the TLS handshake with the receiver including the check of the receiver's TLS certificate.

190 of 287

Check availability status of receiver

Find out the state of the receiver service:

- Available: proceed and route the message
- Unavailable: provide (asynchronous) message for store & forward

Determine asynchronous message context

Determine whether the message is sent in a synchronous or asynchronous context. This is configured inside the CRE for each operation to be called for the message. Here, we assume that the asynchronous context was configured in the operation definition.

CRE::Store asynchronous message

See [CRE::Store asynchronous message](#).

Send delivery acknowledgement to the sender

Send a [deliveryAcknowledgement](#) to the sender with the value **CRE** from [DeliveredToEnum](#).

Handle connection error

If the connection cannot be established (TLS problems, connect timeout, etc.), trigger message queuing and a [deliveryAcknowledgement](#) response to the sender.

CRE::Send asynchronous message to the receiver

See [CRE::Send asynchronous message to the receiver](#).

Pass receiver output to the sender

The receiver output is sent to the sender. This output can be the specified [deliveryRejection](#) or any other output (not further specified here).

Examine communication error

Checks the incoming error content and decide the further processing.

- message potentially re-deliverable later
If the content is an instance of [deliveryRejection](#) with error identifier [E IONC RECEIVER RESPONSE TIMEOUT](#), [E IONC UNKNOWN OPERATION](#) or [E IONC UNKNOWN TECHNICAL PROBLEM](#)
- no messages deliverable now
If the content is an HTTP 503 or an instance of [deliveryRejection](#) with error identifier [E IONC NO TARGET ADDRESS FOUND FOR ROUTING PARAMETERS](#) or [E IONC RECEIVER NOT REACHABLE](#)
- message not re-deliverable
otherwise, e.g.
- any further [deliveryRejection](#) code which is not in the list from above

- any further SOAP faults with HTTP 500,
- any other unexpected reaction of the recipient system

CRE::Receive message

See [CRE::Receive message](#).

Convert to delivery rejection

Embed the incoming [EventIdentifier](#) into a [deliveryRejection](#) with the specified format in [Asynchronous Reply Overview](#).

6.2.4 Store & Forward

Contains use cases which are relevant for queuing messages and implementing a store & forward functionality in the CRE.

To implement the ION specification for a back-office system, the content of this chapter is not mandatory. But it may help to understand the message queuing inside the CRE, especially if an error occurs.

6.2.4.1 Supporting objects

Contains datastores that are referenced in the use cases of [Store & Forward](#).

6.2.4.1.1 Dead letter queue

The dead letter queue contains messages that are definitely undeliverable.
The typical reasons are:

- the WSS message expiry date has been reached
- certain [deliveryRejection](#) codes, e.g. duplicate message or similar
- any undefined SOAP faults with HTTP 500
- any other unexpected reply of the recipient system

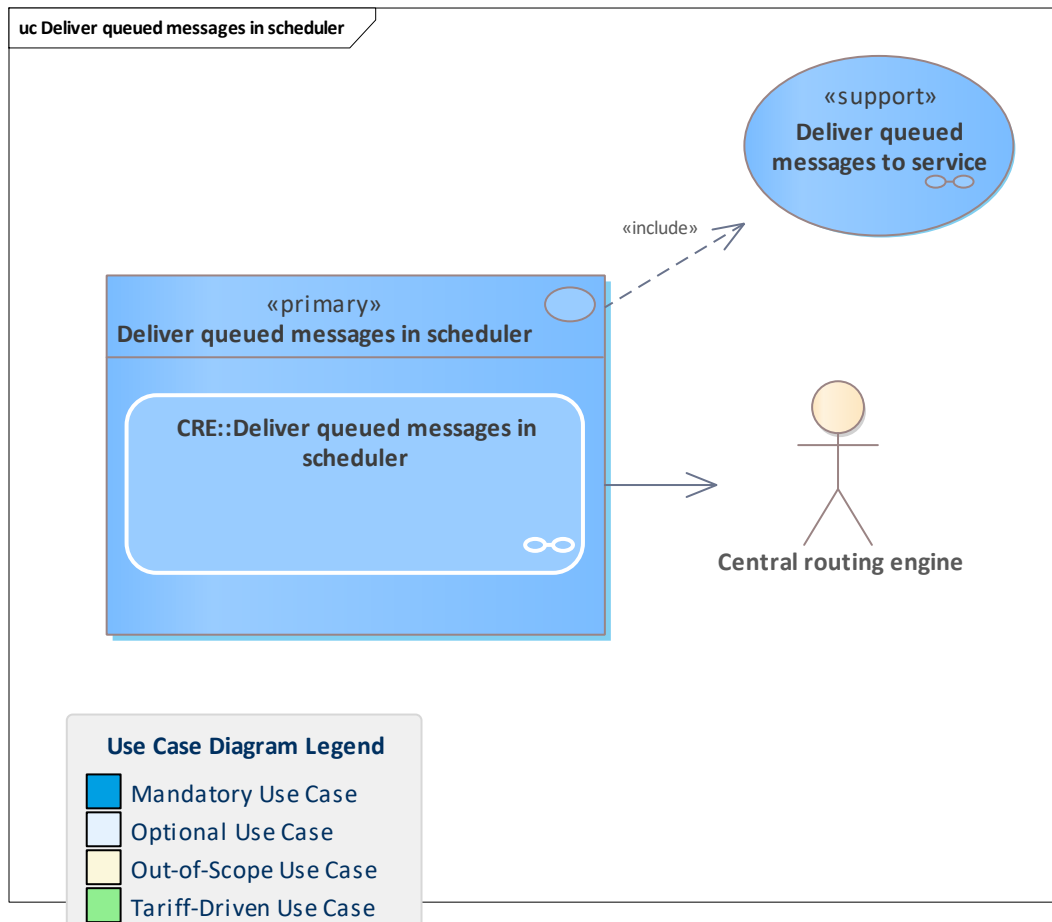
Messages are stored for a certain time span defined in the [\[5\] SLAs for Message Transfer](#).

6.2.4.1.2 Redelivery queue

Persistent queue that stores [Temporarily storable ION messages](#) which cannot be delivered for certain reasons.

Each message is stored together with its metadata that allows filtering per service or recipient. The redeliver queue ensures that messages are queued in their correct sequence when they enter the CRE and stored in the queue.

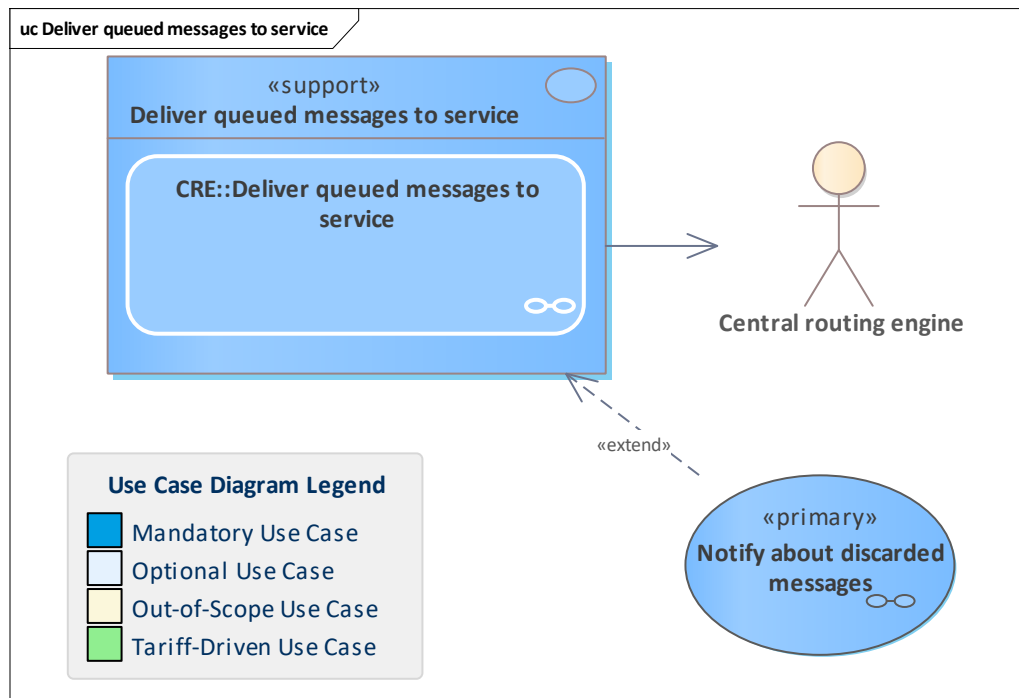
6.2.4.2 Deliver queued messages in scheduler



Use Case	Deliver queued messages in scheduler
Description	<p>Use case frequently started in a scheduler (see [3] CRE Specification: Message Store and Forward) which delivers one or more messages which were previously stored in the queue.</p> <p>For any reason, a participant and its services can be announced as available even in cases where communication problems might have occurred, resulting in the temporary storage of messages. Thus, a scheduled process is needed to determine, if any messages have to be delivered for each service announced as available by each active participant.</p> <p>Note: by considering only active participants, messages of deactivated senders and receivers remain in the queue until there have been activated again or the messages expire and are moved to the dead letter queue.</p>
Initiating Actor	
Reacting Actor	Central routing engine
Preconditions	
Postconditions	
Linked Use Cases (Extended By)	
Linked Use Cases (Includes)	Deliver queued messages to service
Linked Use Cases (Realises)	
Base Activity	
Inputs	Interface version

See [CRE::Deliver queued messages to service](#).

6.2.4.3 Deliver queued messages to service

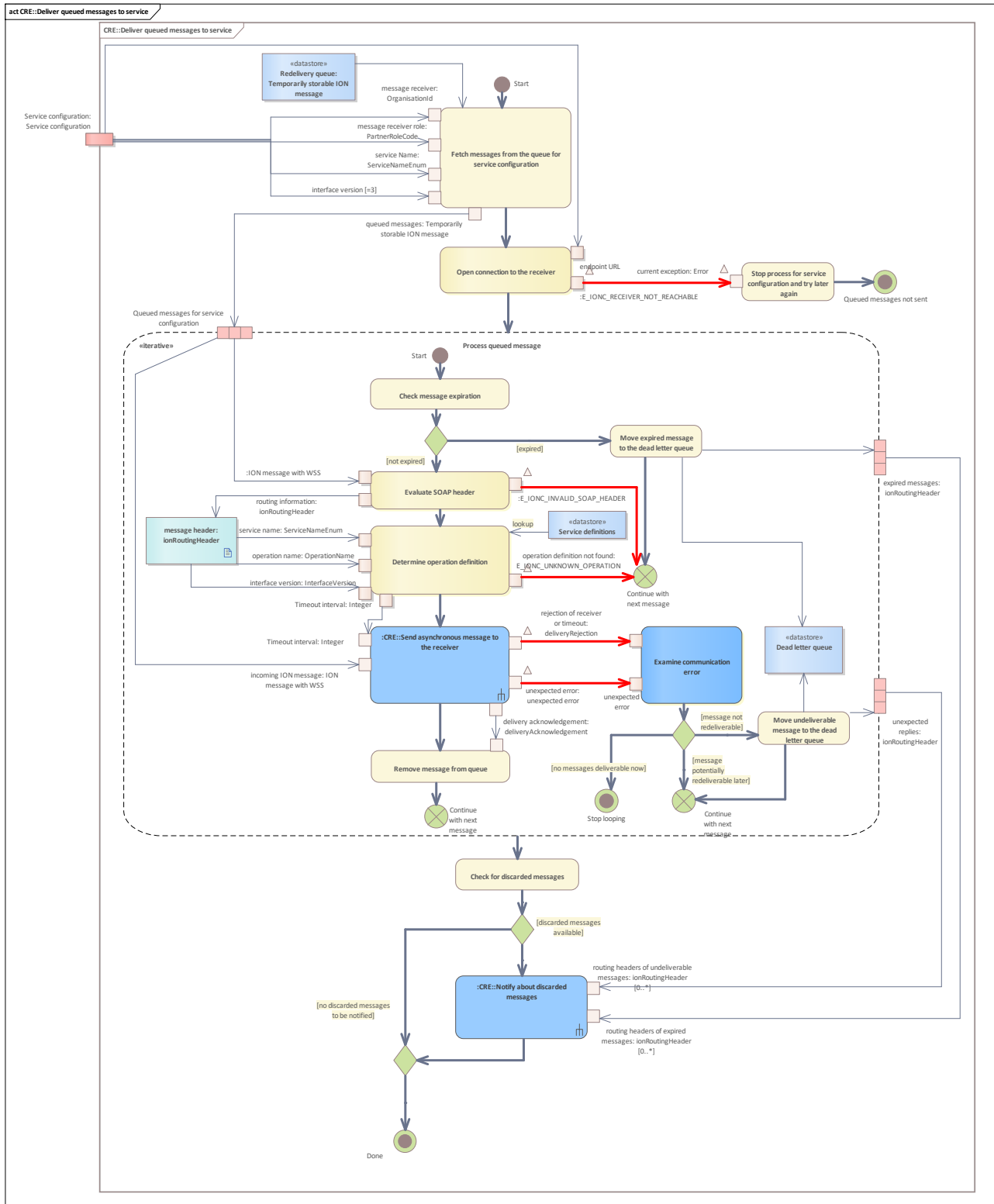


Use Case	Deliver queued messages to service
Description	Supporting use case that delivers queued ION messages to the endpoint URL of a defined service configuration. Collects also potential discarded messages information about expired messages or messages that ultimately cannot be delivered to the intended receiver.
Initiating Actor	
Reacting Actor	Central routing engine
Preconditions	
Postconditions	
Linked Use Cases (Extended By)	Notify about discarded messages
Linked Use Cases (Includes)	
Linked Use Cases (Realises)	
Base Activity	
Inputs	Service configuration : Service configuration
Outputs	
Error Cases	
Activity Diagram	CRE::Deliver queued messages to service

6.2.4.3.1 CRE::Deliver queued messages to service

Activity for [Deliver queued messages to service](#).

6.2.4.3.1.1 CRE::Deliver queued messages to service



Activity diagram for [Deliver queued messages to service](#).

Fetch messages from the queue for service configuration

Does a lookup in the re-delivery queue for messages, which metadata contains

- the organisation ID of the message receiver
- the role of the message receiver
- the service of the message receiver
- the interface version

This information is extracted from the passed [Service configuration](#) and compared with the metadata of the messages in the queue.

Open connection to the receiver

Opens the connection and do the TLS handshake with the receiver including the check of the receiver's TLS certificate.

If a communication problem occurs, an [E IONC RECEIVER NOT REACHABLE](#) occurs.

Determine operation definition

The operation context can be determined by

- the operation name
- the service name
- the interface version

Note: the [InterfaceVersion](#) has to be parsed. Only the major version is relevant for the routing parameters. The string of the interface version follows the rules of <https://semver.org/>

The operation definition consists of the following information (including the information from above):

- the parent service (with the service name from above) that provides this operation
- the operation context (synchronous or asynchronous)
- the operation timeout

If no operation context can be found, an [E IONC UNKNOWN OPERATION](#) occurs.

Stop process for service configuration and try later again

The service which was specified in the [Service configuration](#) element cannot be reached for any reason.

Stop for now and do a retry mechanism, as specified in [\[3\] CRE Specification: Message Store and Forward](#), until the message's WSS expiry timestamp is reached.

CRE::Send asynchronous message to the receiver

See [CRE::Send asynchronous message to the receiver](#).

Remove message from queue

Remove the message from the re-delivery queue and log the acknowledgement of the receiver.

Examine communication error

Checks the incoming error content and decide the further processing.

- message potentially re-deliverable later
If the content is an instance of [deliveryRejection](#) with error identifier [E IONC RECEIVER RESPONSE TIMEOUT](#), [E IONC UNKNOWN OPERATION](#) or [E ION UNKNOWN TECHNICAL PROBLEM](#)
- no messages deliverable now
If the content is an HTTP 503 or an instance of [deliveryRejection](#) with error identifier [E IONC NO TARGET ADDRESS FOUND FOR ROUTING PARAMETERS](#) or [E IONC RECEIVER NOT REACHABLE](#)
- message not re-deliverable otherwise, e.g.
 - any further [deliveryRejection](#) code which is not in the list from above
 - any further SOAP faults with HTTP 500,
 - any other unexpected reaction of the recipient system

Check message expiration

Check if the time stamp of the message is still within in the maximal allowed delivery time span. The time span is defined in [\[5\] SLAs for Message Transfer](#).

Move expired message to the dead letter queue

The expired message is moved from the redelivery queue to the dead letter queue.
The entry contains

- The original message
- The message metadata for later lookup purposes

Add the message (header) also to the list of expired messages

Move undeliverable message to the dead letter queue

The undeliverable message is moved from the redelivery queue to the dead letter queue.
The entry contains

- The original message
- The message metadata for later lookup purposes

Add the message (header) to the list of unexpected replies.

CRE::Notify about discarded messages

See [CRE::Notify about discarded messages](#).

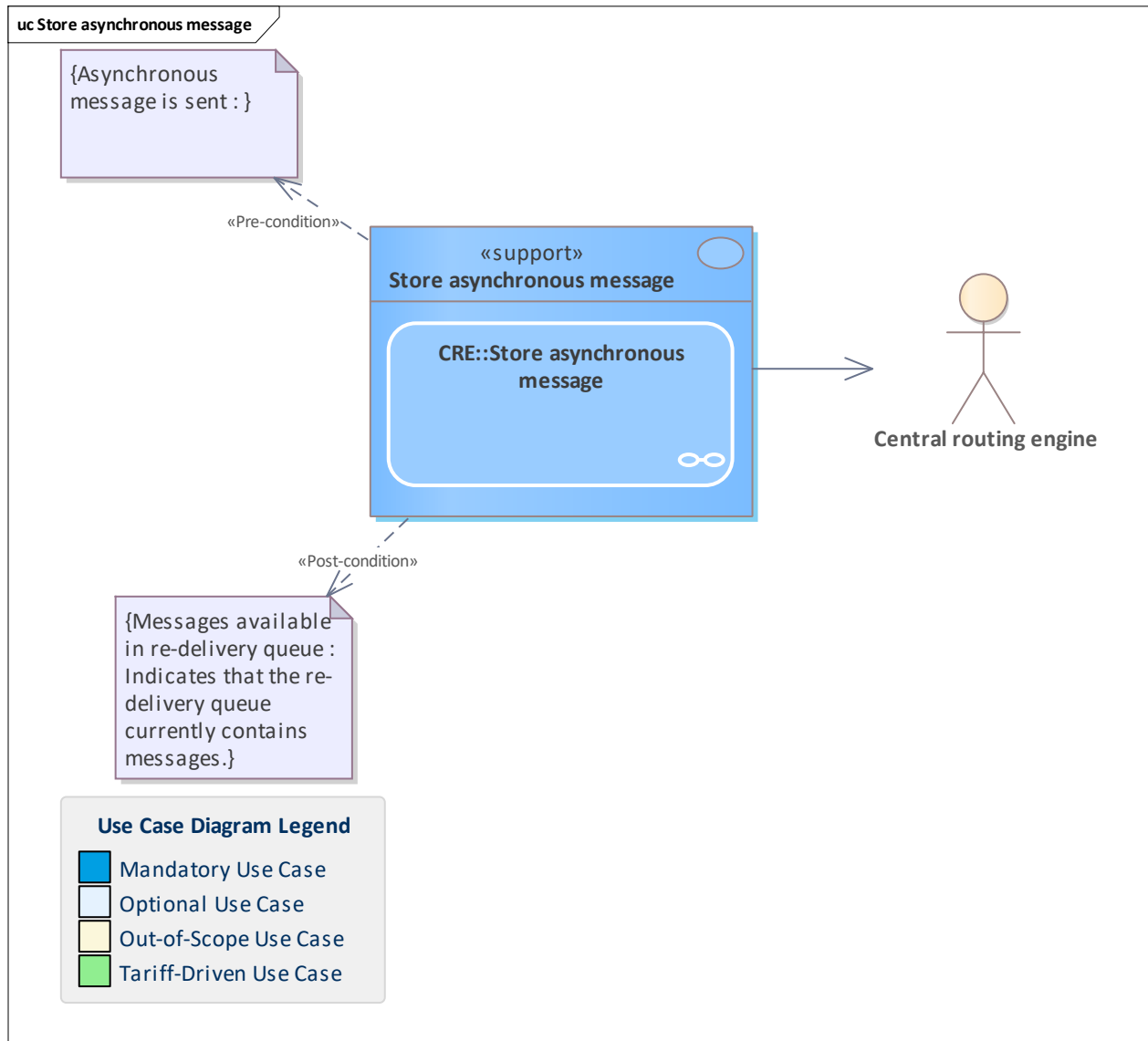
Check for discarded messages

[Notify about discarded messages](#) only if at least one of the passed lists contains at least one element.

Evaluate SOAP header

Checks that the SOAP header has the defined format and parse the fields for routing purposes.

6.2.4.4 Store asynchronous message



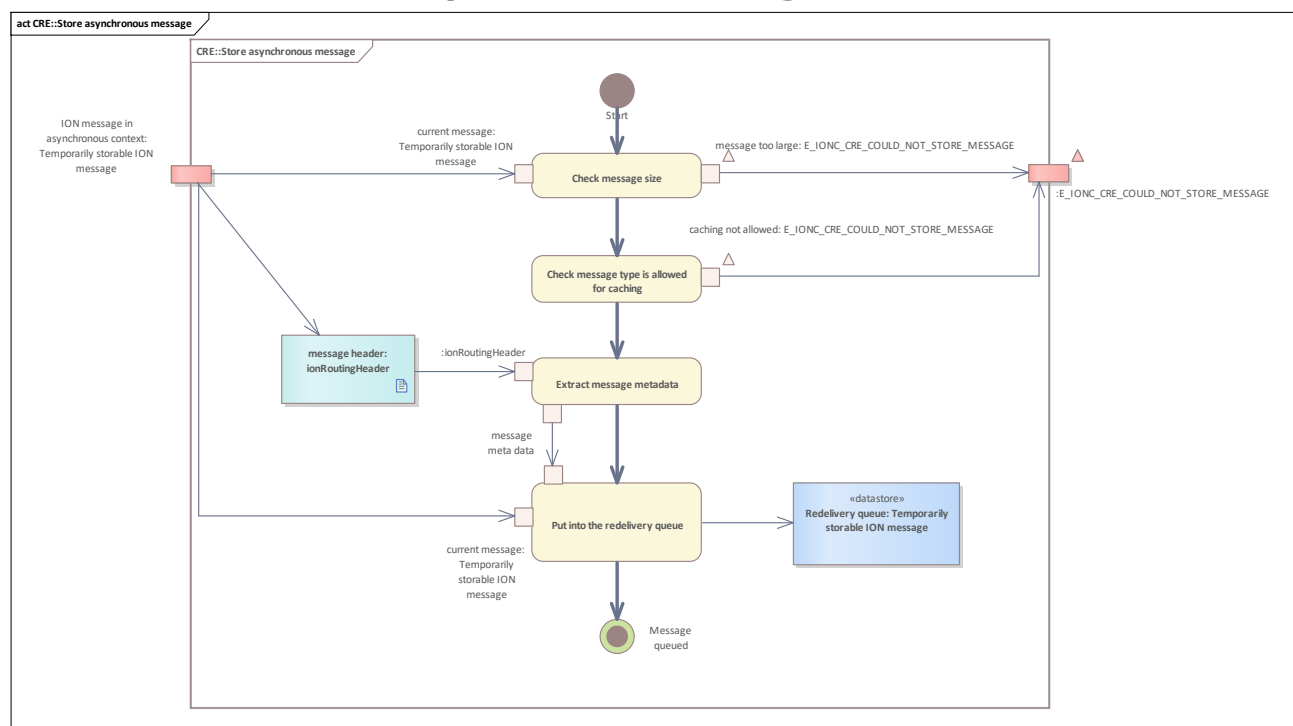
Use Case	Store asynchronous message
Description	<p>Supporting use case to store a message temporarily in an asynchronous context. Queuing is only possible if the message is configured as asynchronous in the CRE. In this case, the CRE may store the message in certain scenarios:</p> <ul style="list-style-type: none"> • The receiving system's service is not set to available • The receiving system's service is announced as available but the response HTTP code is 503 (temporary not available) <p>In these cases, the message is put into a queue for later delivery.</p>
Initiating Actor	
Reacting Actor	Central routing engine

Preconditions	Asynchronous message is sent
Postconditions	Messages available in re-delivery queue
Linked Use Cases (Extended By)	
Linked Use Cases (Includes)	
Linked Use Cases (Realises)	
Base Activity	
Inputs	ION message in asynchronous context : Temporarily storable ION message
Outputs	
Error Cases	E IONC CRE COULD NOT STORE MESSAGE E IONC CRE COULD NOT STORE MESSAGE E IONC CRE COULD NOT STORE MESSAGE: E IONC CRE COULD NOT STORE MESSAGE
Activity Diagram	CRE::Store asynchronous message

6.2.4.4.1 CRE::Store asynchronous message

Activity for [Store asynchronous message](#).

6.2.4.4.1.1 CRE::Store asynchronous message



Activity diagram for [Store asynchronous message](#).

Check message size

Check the size of the incoming message. The size must not be larger than the configured threshold value. See [Maximum message size](#).

Check message type is allowed for caching

The CRE has a configured list with operations for which related requests must not be stored and forwarded. This can be configured for certain use cases with sensitive data.

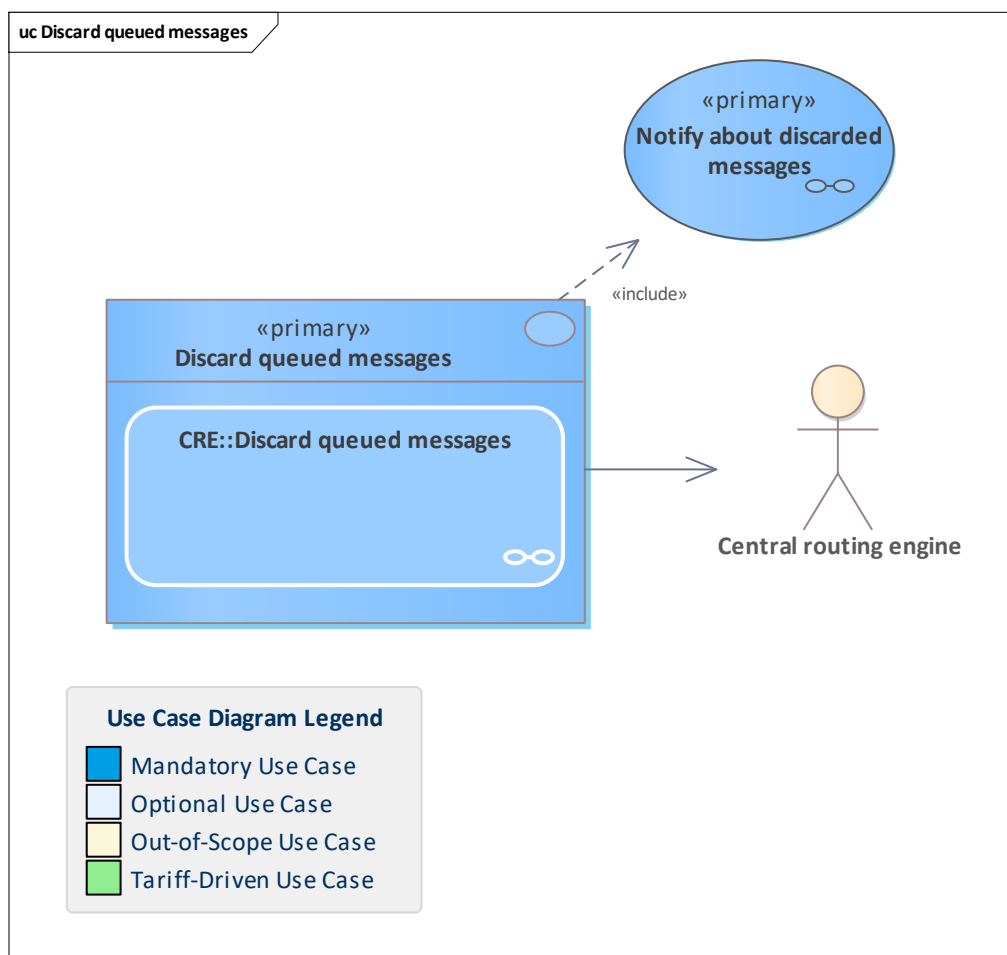
Put into the redelivery queue

Put the entire message into the queue together with its metadata to be accessed later. The aim is to find the message and send it unchanged to the receiver at a later time.

Extract message metadata

Extracts the metadata of the message for later delivery purposes. Using this data, the message can be found in the queue.

6.2.4.5 Discard queued messages



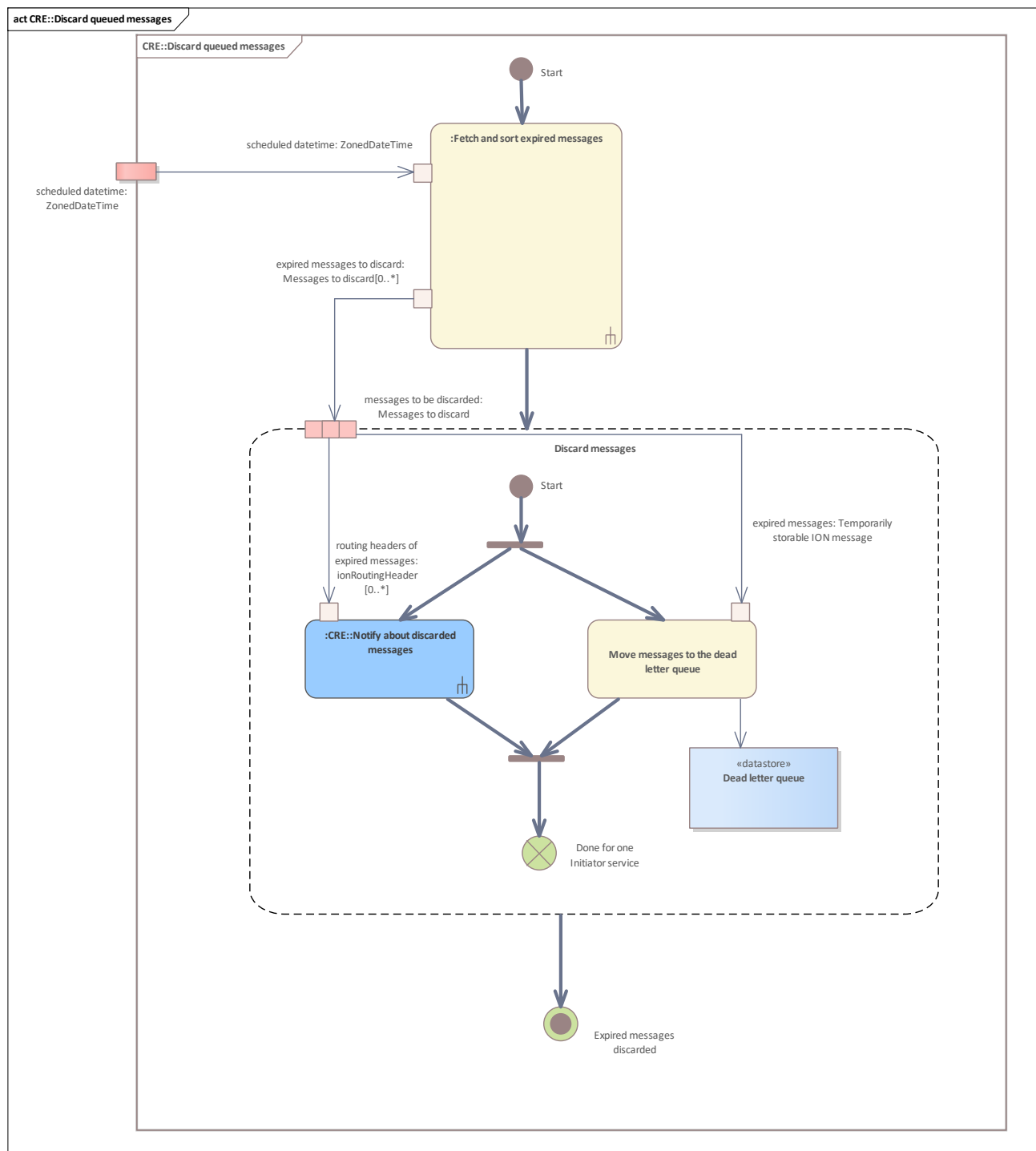
Use Case	Discard queued messages
Description	Scheduled process which frequently looks up for expired messages in the re-delivery queue. The final attempt of (configured) delivery iterations failed. Messages become expired due to the WSS expiration timestamp defined in [5] SLAs for Message Transfer . The messages are moved to the dead letter queue and the meta-information of the messages (ionRoutingHeader elements) is used

	to build a notifyDiscardedMessages element which is sent to the original sender.
Initiating Actor	
Reacting Actor	Central routing engine
Preconditions	
Postconditions	
Linked Use Cases (Extended By)	
Linked Use Cases (Includes)	Notify about discarded messages
Linked Use Cases (Realises)	
Base Activity	
Inputs	scheduled datetime : ZonedDateTime
Outputs	
Error Cases	
Activity Diagram	CRE::Discard queued messages

6.2.4.5.1 CRE::Discard queued messages

Activity for [Discard queued messages](#).

6.2.4.5.1.1 CRE::Discard queued messages



Activity diagram for [Discard queued messages](#).

CRE::Notify about discarded messages

See [CRE::Notify about discarded messages](#).

Move messages to the dead letter queue

In case of any delivery error, the messages are moved to the dead letter queue.

Each entry contains

- The original message
- The message metadata for later lookup purposes

Fetch and sort expired messages

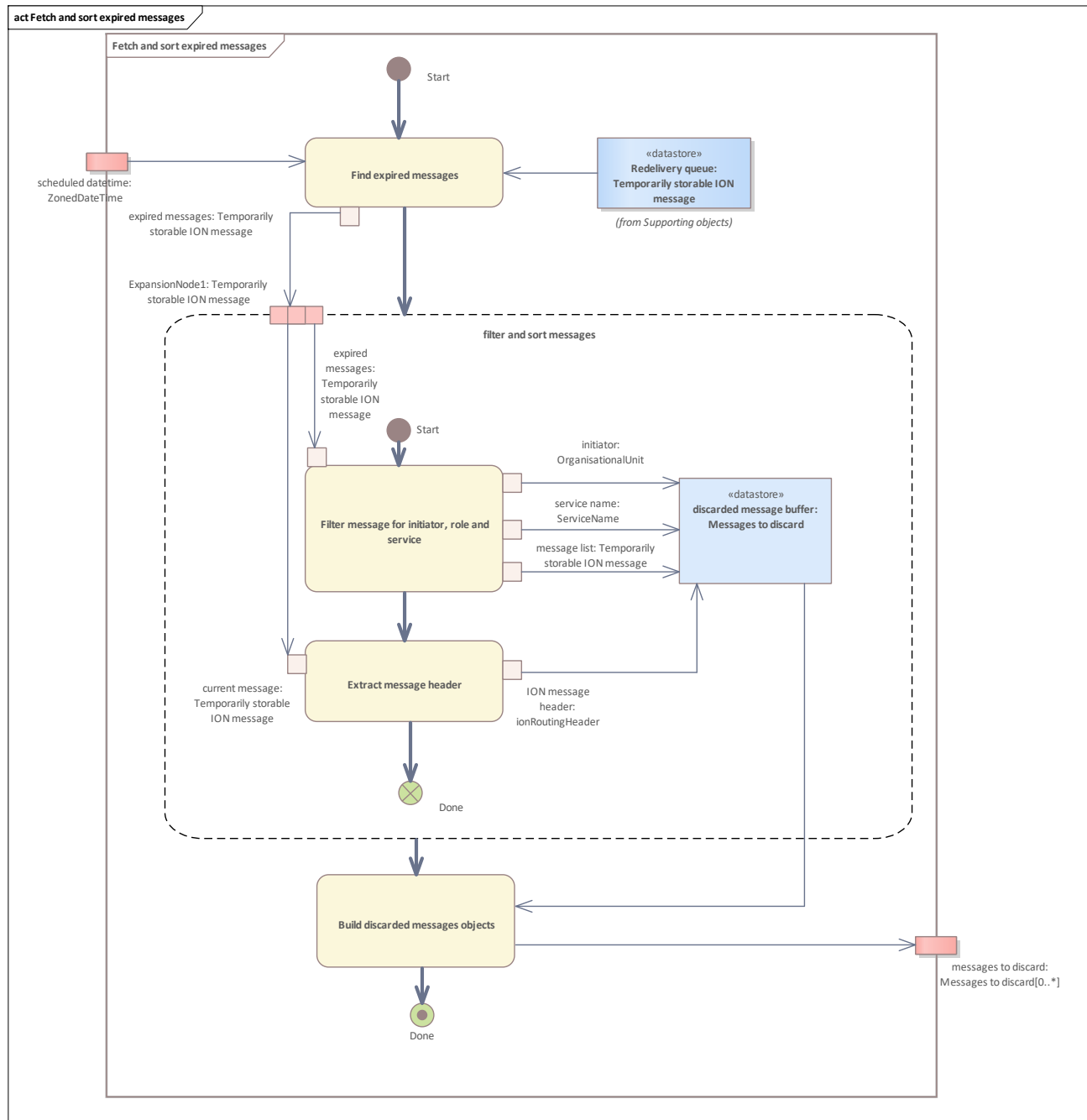
See [Fetch and sort expired messages](#).

6.2.4.5.1.2 Fetch and sort expired messages

Does a lookup in the re-delivery queue for messages, which header's WSS expiration timestamp is equal to or less than the current timestamp. These messages can no be longer provided for re-delivery and have to be moved to the dead letter queue.

Provides these messages for the dead letter queue and the set of [ionRoutingHeader](#) objects of these messages for [Notify about discarded messages](#).

6.2.4.5.1.2.1 Fetch and sort expired messages



Extract message header

Extract the [ionRoutingHeader](#) from the expired message.

Filter message for initiator, role and service

Filter all expired messages and build a map containing

- the initiator service (organisation ID, role and service name) as key
- the list of expired messages for this service
- the list of message headers for these expired messages

Build discarded messages objects

Takes the [Discarded messages](#) objects from the datastore and provide them as activity parameter.

Find expired messages

Filters and lists the expired messages by the passed timestamp coming from the [Redelivery queue](#).

6.2.4.6 Messages expired

The final attempt of (configured) delivery iterations failed. Messages become expired due to the WSS expiration timestamp defined in [\[5\] SLAs for Message Transfer](#).

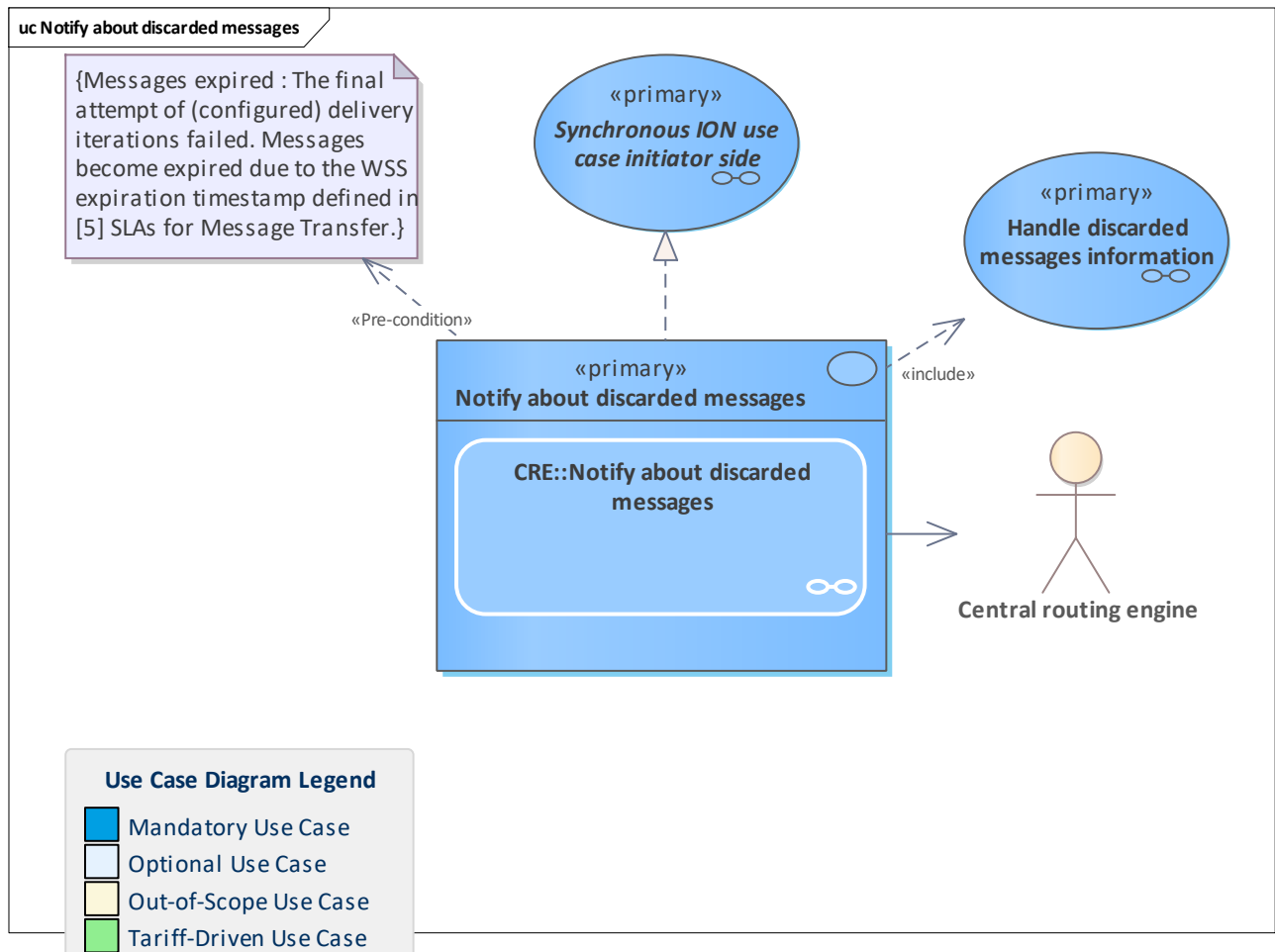
6.2.4.7 Messages available in re-delivery queue

Indicates that the re-delivery queue currently contains messages.

6.2.5 Monitoring and notification

Contains use cases for monitoring in the CRE especially in the store & forward context. This chapter is not mandatory for back-office system implementation, but it will help to understand and to implement the use case [Handle discarded messages information](#).

6.2.5.1 Notify about discarded messages



Use Case	Notify about discarded messages
Description	<p>Messages in the Redelivery queue can be discarded for a specific sender service (organisation ID, role ID and service name) when the use cases Deliver queued messages to service (triggered by scheduler or when setting a service to available status) or Discard queued messages are executed. Accordingly, this use case is called in these use cases.</p> <p>Group these messages for the sender service by recipient, recipient role, service and message type (operation name) as well as the number of messages in this group. Then send this information in the form of DiscardedMessages back to the sender which had previously sent these messages to the intended, but not reached, recipient.</p> <p>Distinguish between expired messages and undeliverable messages.</p>
Initiating Actor	
Reacting Actor	Central routing engine
Preconditions	Messages expired
Postconditions	
Linked Use Cases (Extended By)	
Linked Use Cases (Includes)	Handle discarded messages information
Linked Use Cases (Realises)	Synchronous ION use case initiator side
Base Activity	

Inputs	routing headers of undeliverable messages : ionRoutingHeader routing headers of expired messages : ionRoutingHeader
Outputs	
Error Cases	
Activity Diagram	CRE::Notify about discarded messages

6.2.5.1.1 CRE::Notify about discarded messages

Activity for [Notify about discarded messages](#).

- Receiver and receiver role and service (where the messages could not be delivered to)
- Message type using the original operation name

Provides additionally the information (for the grouped messages of above)

- The timestamp of the oldest message involved
- The timestamp of the newest message involved
- The number of expired and undeliverable messages

To avoid message broadcasting due to thousands of existing messages in the queue, only the described meta data from above is collected and later sent, not the messages themselves.

Create request

Builds the request with element [notifyDiscardedMessages](#).

Sends the request to the original sender of the collected messages (sender consists of organisation ID, role, service).

BO-*::Handle discarded messages information

See [BO-*::Handle discarded messages information](#).

Update status in the initiator system

A business response or exception is received and status is updated accordingly in the initiator's system.

The generic message correlation steps are done before in the use cases [Receive and validate asynchronous ION response](#), [Receive and validate asynchronous business exception](#), [Receive and validate synchronous ION response](#) or [Receive and validate synchronous ION business exception](#).

This step should update the status of a certain entity depending on the underlying use case (e.g. set the state of an entitlement to "hotlisted").

In the case of a business exception: depending on exception type, a subsequent handling may be required after updating the status to enable regular processing in further actions. **It is assumed that the problem defined in the exception is solved before the action completes. Only then, the follow-up steps can continue.**

In the case of warnings contained in the reply, a further warning handling could be necessary. If the [response payload](#) contains [Response business data](#), this business data must be processed in further steps.

BO-Main::Handle process instance ID

See [BO-Main::Handle process instance ID](#).

6.2.6 Supporting activities

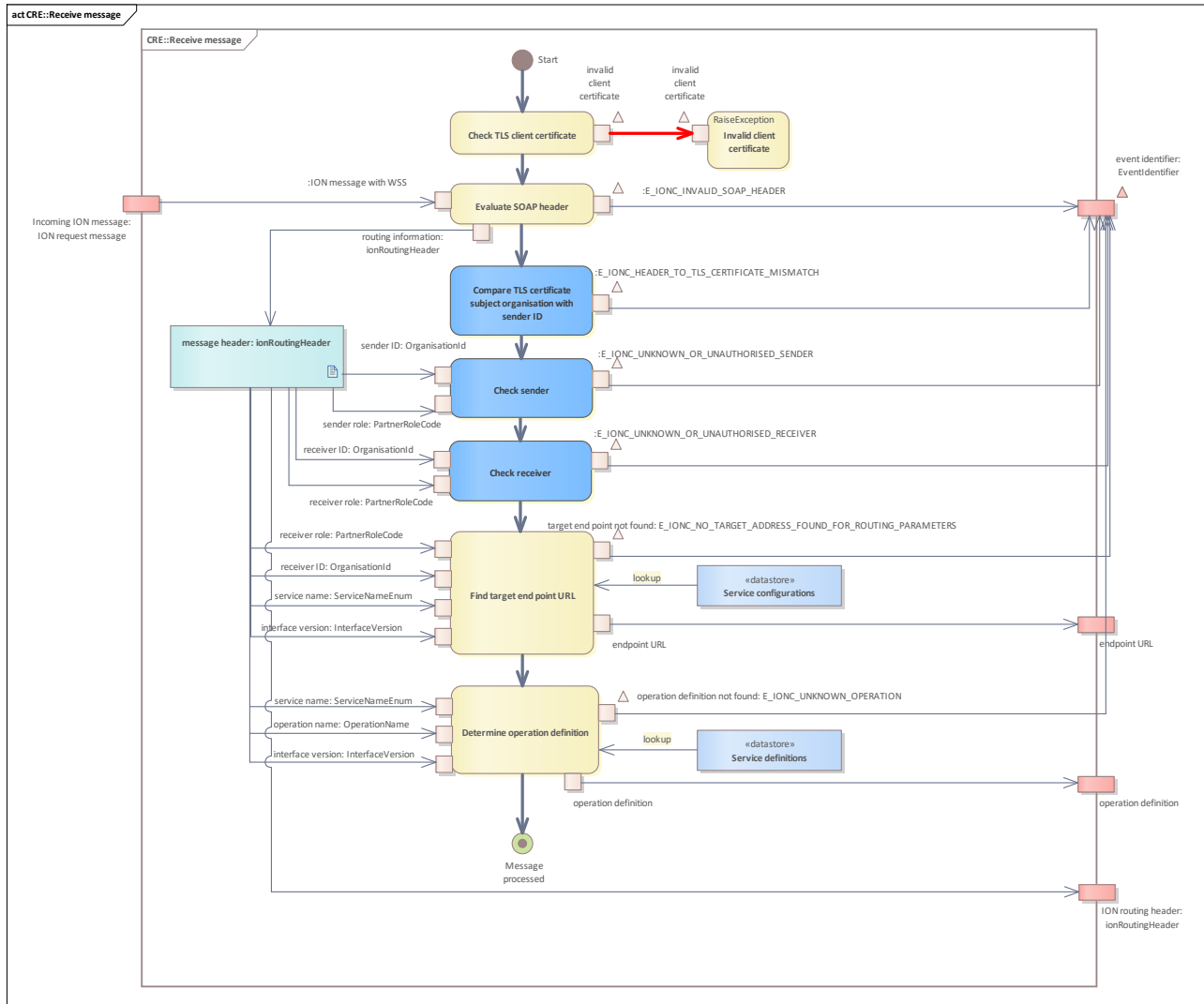
This package contains activities that are needed in more than one use case for ION message routing in the CRE.

6.2.6.1 CRE::Receive message

The CRE receives a message either for routing or for queuing if the receiver server is not available. General checks are performed and the routing information is extracted (URL, operation timeout, etc.)

The activity is the same for [Process synchronous message](#) and [Process asynchronous message](#).

6.2.6.1.1 CRE::Receive message



Activity diagram for [CRE::Receive message](#).

Determine operation definition

The operation context can be determined by

- the operation name
- the service name
- the interface version

Note: the [InterfaceVersion](#) has to be parsed. Only the major version is relevant for the routing parameters. The string of the interface version follows the rules of <https://semver.org/>

The operation definition consists of the following information (including the information from above):



- the parent service (with the service name from above) that provides this operation
- the operation context (synchronous or asynchronous)
- the operation timeout

If no operation context can be found, an [E IONC UNKNOWN OPERATION](#) occurs.

Find target end point URL

The CRE has to determine the configured target end point URL. This URL is found by the parameters

- sender ID
- sender role
- sender service
- interface version

Note: the [InterfaceVersion](#) has to be parsed. Only the major version is relevant for the routing parameters. The string of the interface version follows the rules of <https://semver.org/>. If no URL can be found for these parameters in the CRE configuration, an [E IONC NO TARGET ADDRESS FOUND FOR ROUTING PARAMETERS](#) exception occurs.

Check receiver

Check if the receiver is unknown or not authorised.

In this context, receiver means the combination of organisation ID und role.

- CRE must have the receiver in its master data
- CRE checks that the receiver is active

The active status is linked to an approved successful functional test of the receiver's system.

Check sender

Checks if the sender is unknown or not authorised.

In this context, sender means the combination of organisation ID und role.

- CRE must have the sender in its master data
- CRE checks that the sender is active

The active status is linked to an approved successful functional test of the sender's system.

Compare TLS certificate subject organisation with sender ID

Extracts the subject from the TLS certificate and determine the organisation ID from the field x509.subject.organization ("O"). Compares it with the organisation ID of the sender stored in [ionRoutingHeader](#).

In case of mismatch, does a further check to see if a joint service broker (JSB) is employed.

This is configured in the CRE. If the organisation ID in the certificate's subject matches a configured joint service broker organisation ID and if the sender organisation ID is configured to be served by this JSB, the message will be processed.

The message is rejected otherwise with an

[E IONC HEADER TO TLS CERTIFICATE MISMATCH](#).

Evaluate SOAP header

~~Checks that the SOAP header has the defined format and parse the fields for routing purposes.~~

Invalid client certificate

An exception that occurs if the client certificate is not valid.

- if the authority is wrong (not issued by the required (sub-) root authorization)
- the certificate is expired
- the certificate does not belong to the client

The message is rejected with an exception or the socket is closed.

Check TLS client certificate

The sender has to provide its TLS certificate.

All certificates for ION message exchange are issued by one defined authority and have a certain validity.

To verify the client certificate, an OCSP call has to be made to the trust centre or a current certificate revocation list has to be evaluated. Furthermore, the complete certificate chain has to be evaluated. Each certificate of the chain must be valid.

Normally, this is performed in the upstream web server of the receiving system.

See also [Transport encryption via TLS](#) and [Validity of Certificates](#).

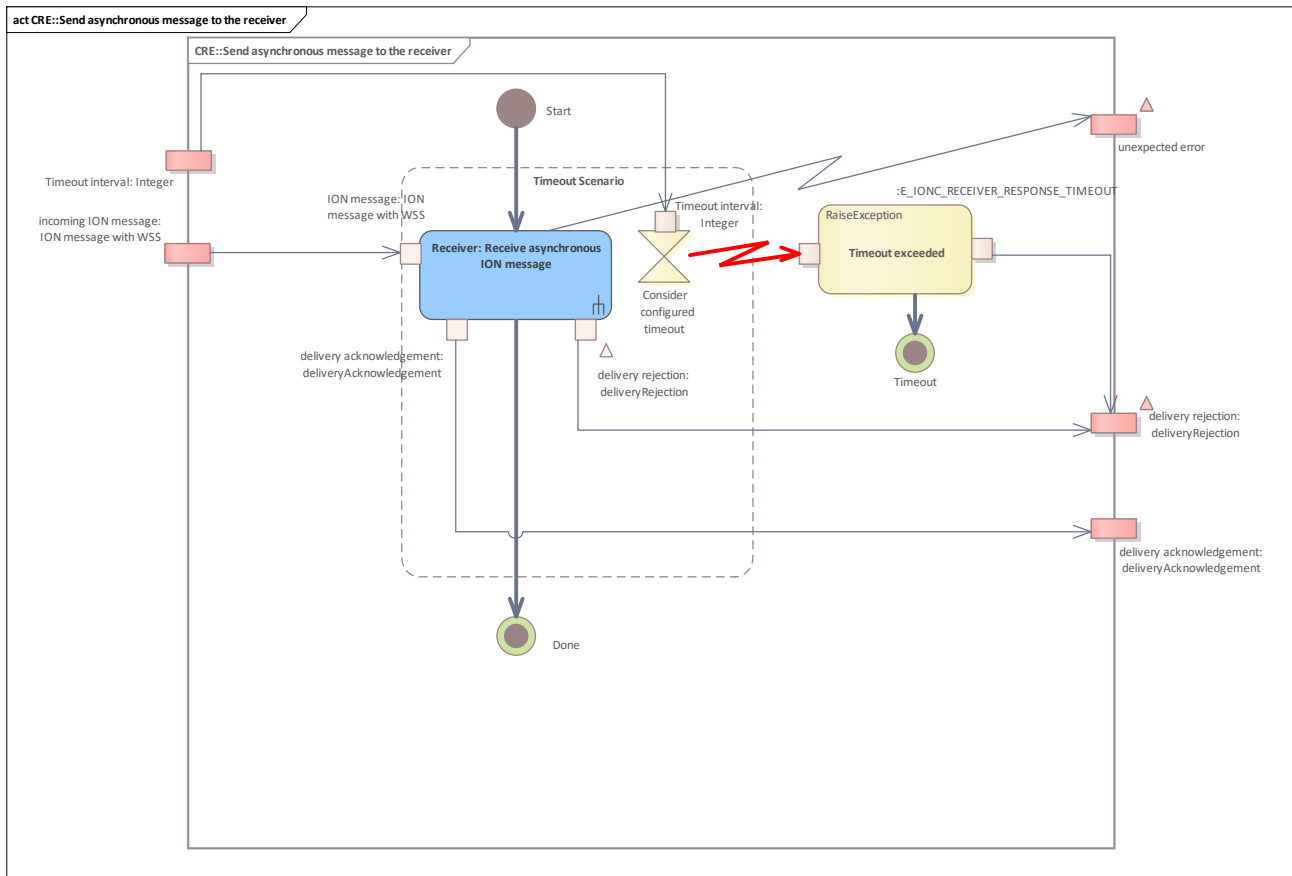
Additionally, this check has to ensure that the right security level is used, see also [Security Levels](#).

Note: If the receiving party is a participant system or a JSB, the TLS client certificate should be always the one of the CRE. If the receiving party is the CRE, it has to verify the TLS client certificate of the participant system or the JSB.

6.2.6.2 CRE::Send asynchronous message to the receiver

Shared activity to send an ION message in asynchronous context to the receiver. Depending on the "direction" in the current asynchronous use case, the receiver can either be the [Processor](#) (Outbound) or the [Initiator](#) (return path).

6.2.6.2.1 CRE::Send asynchronous message to the receiver



Activity diagram for [Process message in asynchronous context](#).

Timeout exceeded

Raises an exception that the configured timeout has been exceeded. This results in an [E_IONC_RECEIVER_RESPONSE_TIMEOUT](#) towards the sender. For asynchronous contexts, the message is **not queued** in the CRE.

Consider configured timeout

For each operation - independent from synchronous or asynchronous context - a timeout is configured inside the CRE by which the accessed system has to respond. The default value can be found in [Appendix: CRE Timeout Configuration Table](#).

Receive asynchronous ION message

See [Receive asynchronous ION message](#).

6.2.7 Supporting objects

Contains CRE use case specific supporting objects like data stores, etc.

6.2.7.1 Service configuration

Conceptual service configuration in the CRE consisting of

- Organisation ID
- Role
- Service name
- Interface version (only major version)
- Endpoint URL

Attributes
endPointURL:
Multiplicity:
interfaceVersion:InterfaceVersion
Multiplicity:
organisationId:OrganisationId
Multiplicity:
role:PartnerRoleCode
Multiplicity:
serviceName:ServiceName
Multiplicity:

6.2.7.2 Messages to discard

Conceptual class containing the organisation ID, the role ID and the service name of the initiator together with a list of messages which are about to be discarded, together with the extracted message headers.

Attributes
initiatorUnit:OrganisationalUnit
Multiplicity:

listOfDiscardedMessageHeaders:ionRoutingHeader
Multiplicity: [1..*]
listOfDiscardedMessages:Temporarily storable ION message
Multiplicity: [1..*]
serviceName:ServiceName
Multiplicity:

6.2.7.3 Service configurations

Stores the routing information for each participant, consisting of

- organisation ID
- role
- service name
- interface version
- endpoint URL

This data is configured by the public transport company via ESH using an online interface.

6.2.7.4 Service definitions

Stores common information about a service. This is part of the master data of the CRE:

- service name
- interface version (only major version)
- security level
- service operations (timeout, synchronous or asynchronous)

6.2.8 Supporting actions

This package contains actions that are needed in more than one use case for the CRE.

6.2.8.1 Check receiver

Check if the receiver is unknown or not authorised.

In this context, receiver means the combination of organisation ID und role.

- CRE must have the receiver in its master data
- CRE checks that the receiver is active

The active status is linked to an approved successful functional test of the receiver's system.

6.2.8.2 Check sender

Checks if the sender is unknown or not authorised.

In this context, sender means the combination of organisation ID und role.

- CRE must have the sender in its master data
- CRE checks that the sender is active

The active status is linked to an approved successful functional test of the sender's system.

6.2.8.3 Check TLS client certificate

The sender has to provide its TLS certificate.

All certificates for ION message exchange are issued by one defined authority and have a certain validity.

To verify the client certificate, an OCSP call has to be made to the trust centre or a current certificate revocation list has to be evaluated. Furthermore, the complete certificate chain has to be evaluated. Each certificate of the chain must be valid.

Normally, this is performed in the upstream web server of the receiving system.

See also [Transport encryption via TLS](#) and [Validity of Certificates](#).

Additionally, this check has to ensure that the right security level is used, see also [Security Levels](#).

Note: If the receiving party is a participant system or a JSB, the TLS client certificate should be always the one of the CRE. If the receiving party is the CRE, it has to verify the TLS client certificate of the participant system or the JSB.

6.2.8.4 Create service configuration

Create or assemble service configuration consisting of

- Interface version (only major version)
- Receiver organisation ID
- Receiver role
- Receiver service
- The related endpoint URL

This configuration can be used to filter and send messages from the re-delivery queue.

6.2.8.5 Compare TLS certificate subject organisation with sender ID

Extracts the subject from the TLS certificate and determine the organisation ID from the field x509.subject.organization ("O"). Compares it with the organisation ID of the sender stored in [ionRoutingHeader](#).

In case of mismatch, does a further check to see if a joint service broker (JSB) is employed.

This is configured in the CRE. If the organisation ID in the certificate's subject matches a configured joint service broker organisation ID and if the sender organisation ID is configured to be served by this JSB, the message will be processed.

The message is rejected otherwise with an

[E IONC HEADER TO TLS CERTIFICATE MISMATCH](#).

6.2.8.6 Determine operation definition

The operation context can be determined by

- the operation name
- the service name
- the interface version

Note: the [InterfaceVersion](#) has to be parsed. Only the major version is relevant for the routing parameters. The string of the interface version follows the rules of <https://semver.org/>

The operation definition consists of the following information (including the information from above):

- the parent service (with the service name from above) that provides this operation
- the operation context (synchronous or asynchronous)
- the operation timeout

If no operation context can be found, an [E IONC UNKNOWN OPERATION](#) occurs.

6.2.8.7 Evaluate SOAP header

Checks that the SOAP header has the defined format and parse the fields for routing purposes.

6.2.8.8 Examine communication error

Checks the incoming error content and decide the further processing.

- message potentially re-deliverable later
If the content is an instance of [deliveryRejection](#) with error identifier [E IONC RECEIVER RESPONSE TIMEOUT](#), [E IONC UNKNOWN OPERATION](#) or [E ION UNKNOWN TECHNICAL PROBLEM](#)
- no messages deliverable now
If the content is an HTTP 503 or an instance of [deliveryRejection](#) with error identifier [E IONC NO TARGET ADDRESS FOUND FOR ROUTING PARAMETERS](#) or [E IONC RECEIVER NOT REACHABLE](#)
- message not re-deliverable
otherwise, e.g.
 - any further [deliveryRejection](#) code which is not in the list from above
 - any further SOAP faults with HTTP 500,
 - any other unexpected reaction of the recipient system

6.2.8.9 Find target end point URL

The CRE has to determine the configured target end point URL. This URL is found by the parameters

- sender ID
- sender role
- sender service
- interface version

Note: the [InterfaceVersion](#) has to be parsed. Only the major version is relevant for the routing parameters. The string of the interface version follows the rules of <https://semver.org/>. If no URL can be found for these parameters in the CRE configuration, an [E_IONC_NO_TARGET_ADDRESS_FOUND_FOR_ROUTING_PARAMETERS](#) exception occurs.

6.2.8.10 Invalid client certificate

An exception that occurs if the client certificate is not valid.

- if the authority is wrong (not issued by the required (sub-) root authorization)
- the certificate is expired
- the certificate does not belong to the client

The message is rejected with an exception or the socket is closed.

6.2.8.11 Open connection to the receiver

Opens the connection and do the TLS handshake with the receiver including the check of the receiver's TLS certificate.

If a communication problem occurs, an [E_IONC_RECEIVER_NOT_REACHABLE](#) occurs.

6.2.8.12 Timeout exceeded

Raises an exception that the configured timeout has been exceeded. This results in an [E_IONC_RECEIVER_RESPONSE_TIMEOUT](#) towards the sender.

For asynchronous contexts, the message is **not queued** in the CRE.

6.2.8.13 Timeout Scenario

Region for the timeout. Wraps the message exchange between the CRE and the recipient of the message.

6.2.8.13.1 Consider configured timeout

For each operation - independent from synchronous or asynchronous context - a timeout is configured inside the CRE by which the accessed system has to respond.

The default value can be found in [Appendix: CRE Timeout Configuration Table](#).

6.3 Supporting objects

This package contains classes, objects and components which are needed in more than one use case or activity and in the CRE as well as for ION messaging.

6.3.1 Keys and certificates

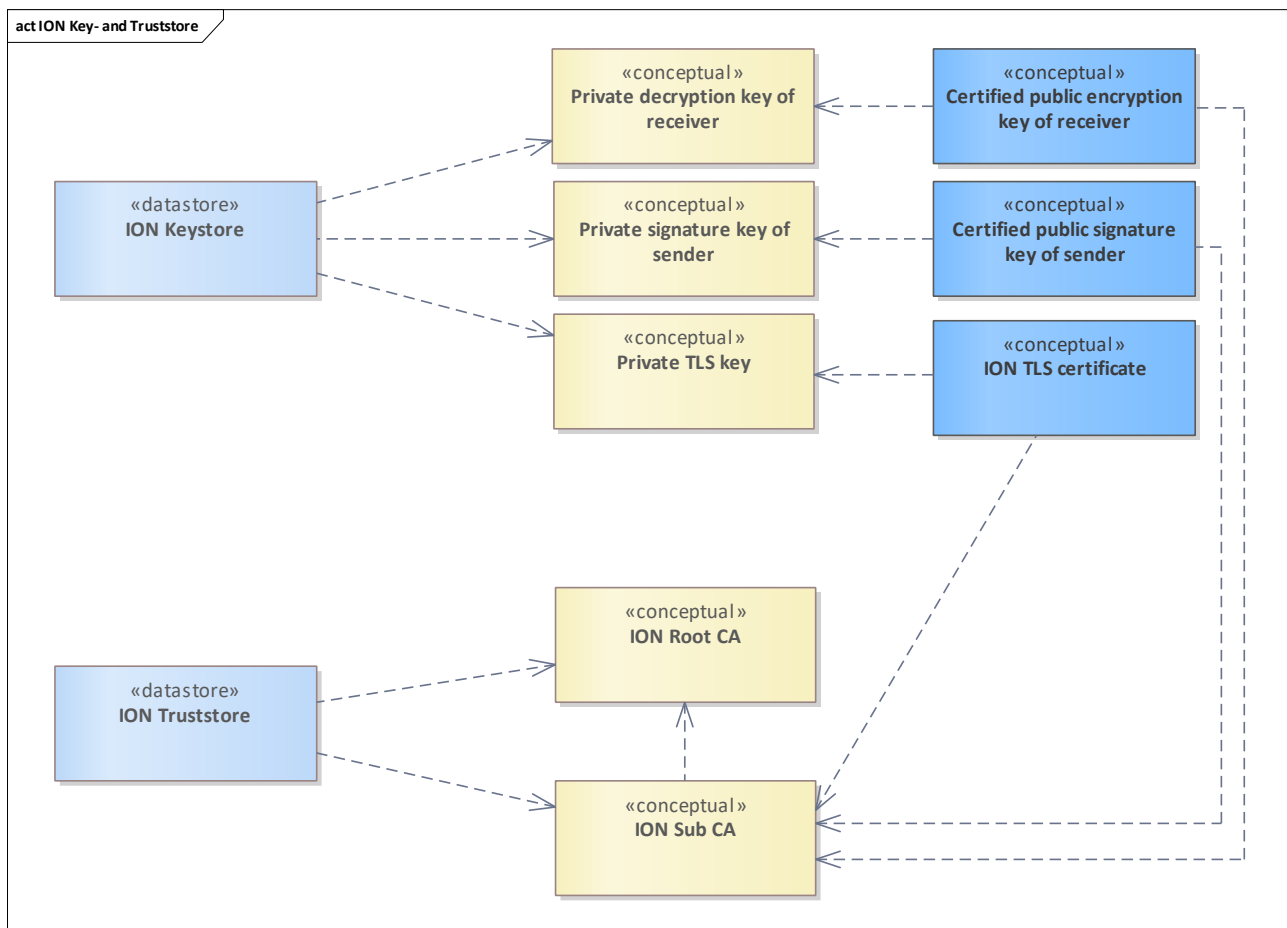


Figure 32: ION Key- and Truststore

6.3.1.1 ION Root CA

X.509 certificate of the ION root CA. The root CA certifies all its sub CAs.

6.3.1.2 ION Sub CA

X.509 certificate of the ION sub CA. The sub CA certifies all ION public keys for TLS, encryption and signature.

6.3.1.3 Private decryption key of receiver

Private key of the receiver of a message. Normally, this key is stored securely in a keystore and is read from there when messages are received.

The receiver is always the [Processor](#) for an initial request. For asynchronous responses, the receiver is the [Initiator](#).

This key is used to decrypt the message. According to the etiCORE WSS policy, the contents of the SOAP body are encrypted. This is generally always the [request payload](#) for initial requests or the [response payload](#) for synchronous and asynchronous responses.

The [Receiver public encryption key](#) associated with the receiver's private key is certified by the PKI, stored in the PKI's central directory and can then be retrieved by the participants via directory services to encrypt the message initially as the sender.

6.3.1.4 Certified public encryption key of receiver

The [Receiver public encryption key](#) associated with the receiver's private key ([Receiver private decryption key](#)) is certified by the PKI and stored in the PKI's central directory and can then be retrieved by the participants via directory services to encrypt the message initially as the sender.

Must be determined/fetched by using the receiver's organisation ID, role and the purpose "encryption".

6.3.1.5 Encryption key reference

Key reference to the asymmetric key that is used for XML encryption in the message, see [Receiver public encryption key](#) and [Receiver private decryption key](#). The key reference is located in the [Security header](#).

The encryption of the [SOAP body](#) and the [Signature](#) was previously performed by a symmetric session key.

This session key has been encrypted by the asymmetric [Receiver public encryption key](#).

To decrypt a message, this session key has to be decrypted first, using the asymmetric [Receiver private decryption key](#). In a second step, the decrypted symmetric session key can be used to decrypt the encrypted XML message parts (in ION the [SOAP body](#) and the [Signature](#)).

6.3.1.6 Private TLS key

Private key for TLS when establishing a connection between client and server.

6.3.1.7 ION TLS certificate

The ION TLS certificate. This is used for client and server authentication during the TLS handshake, depending on the current usage.

6.3.1.8 Private signature key of sender

Private key of the sender of a message. As a rule, this key is stored securely in a keystore and is read from there when messages are sent.

The sender is always the [Initiator](#) for an initial request. For asynchronous responses, the sender is the [Processor](#).

This key is used to sign the message. According to the etiCORE WSS policy, this means signing the contents of the SOAP body. This is generally always the [request payload](#) for initial requests or the [response payload](#) for synchronous and asynchronous responses.

The public key associated with the sender's private key is certified by the PKI and stored in the PKI's central directory. It can then be retrieved by the participants via directory services for signature verification.

6.3.1.9 Certified public signature key of sender

The certified public key of the sender coming from the directory service of the PKI for signature check purposes. The sender had previously used its complement private key to sign the message.

6.3.1.10 Receiver private TLS key

Uses the TLS private key in the receiver context.

6.3.1.11 Receiver server TLS certificate

Used by the client to verify that the server is authentic.

6.3.1.12 Sender private TLS key

Uses the TLS private key in the sender context.

6.3.1.13 Sender client TLS certificate

Used by the server to verify that the client is authentic.

6.3.1.14 ION Keystore

This keystore contains the ION private keys (private-public keys pairs) for message signature (as sender) and message decryption (as receiver) as well as the TLS key that is used for the TLS handshake and TLS authentication on client and server side.

All these key-pairs must be certified by the ION PKI. The certification replies are also imported to this keystore. This import requires at least the existing sub CA certificate stored in the keystore. For better understanding, this is not shown here.

6.3.1.15 ION Truststore

This truststore contains the ION root and sub CA and is used for incoming messages to verify the certificate chain, when leaf certificates of public keys are presented.

6.3.1.16 receiver public key

The current public key of the message receiver. This key is located in the PKI's central directory and must be fetched via the directory service before encrypting a message. It is an instance of [Receiver public encryption key](#).

6.3.1.17 receiver private key

The current private key of the receiver for decryption purposes. It is an instance of [Receiver private decryption key](#) and normally stored in the [ION Keystore](#).

6.3.1.18 sender private key

The current private key of the message sender. Normally, this key is stored securely in the [ION Keystore](#) and is read from there when messages are sent. It is an instance of [Sender private signature key](#).

6.3.1.19 sender public key

The current public key of the sender of a message. Normally, this key has been certified by the trust centre and can be fetched from the directory service of the PKI for signature verification purposes. It is an instance of [Sender public signature key](#). It has to be verified against the [ION Truststore](#) incorporating the CRL.

6.3.2 Message parts and instances

6.3.2.1 SOAP body

Part of the [SOAP Envelope](#) which transports the payload of the ION message. This can be either

- [request payload](#)
- [response payload](#)
- [exception payload](#)

The example uses a message to set a service as available in the CRE as [request payload](#) and is shown without WSS.

The payload in the SOAP body is signed first with the [sender private signature key](#) and then encrypted with the [Receiver public encryption key](#) according to the etiCORE WSS rules.

```
<soap:Body>
  <ns1:setServiceAvailable xmlns="https://eticore.org/ion/3" xmlns:ns1="https://eticore.org/cre/messaging/3">
    <communicationInformation>
      <receiverId>5602</receiverId>
      <receiverRole>254</receiverRole>
      <receiverService>cre</receiverService>
      <messageId>
        <senderId>6598</senderId>
        <senderRole>1</senderRole>
        <senderService>ccp</senderService>
        <messageNumber>4711</messageNumber>
      </messageId>
      <messageTimestamp>2022-08-10T08:53:31.327+02:00</messageTimestamp>
      <!--Optional, set only if greater 0:-->
      <repeatCounter>1</repeatCounter>
      <processInstanceId>SetServiceAvailable_5f7d9b47-b434-416a-9c60-adfde71dec1e</processInstanceId>
    </communicationInformation>
  </ns1:setServiceAvailable>
</soap:Body>
```

6.3.2.2 Encrypted signature

The encrypted signature contains the [Signature](#) part of the [Security header](#).

The encryption of the [Signature](#) was done previously by a symmetric session key.

This session key has been encrypted by the asymmetric [Receiver public encryption key](#). First, it has to be decrypted using the [Receiver private decryption key](#). After that, the [Signature](#) can be decrypted using this symmetric key.

The rule to encrypt the signature is part of the [Appendix: etiCORE Standard-Policy](#).

```
<xenc:EncryptedKey xmlns:xenc=...>
...
  <xenc:CipherData>
    <xenc:CipherValue>
      <!-- contains the encrypted symmetric key used for encryption of signature and body -->
    </xenc:CipherValue>
  </xenc:CipherData>
...
</xenc:EncryptedKey>
```

6.3.2.3 Signature

Contains the signature metadata and its signature value of the plaintext payload embedded in the [SOAP Body](#).

```
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#" Id="SIG-
E379DA17B63A8A4A231615881383064168">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
      <ec:InclusiveNamespaces xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#" PrefixList="soap"/>
    </ds:CanonicalizationMethod>
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <ds:Reference URI="#TS-E379DA17B63A8A4A231615881383049163">
      <ds:Transforms>
        <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
          <ec:InclusiveNamespaces xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#" PrefixList="wsse soap"/>
        </ds:Transform>
      </ds:Transforms>
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue>99GgAficxo97L2VHKJ7n7Tjz7cA=</ds:DigestValue>
    </ds:Reference>
    <ds:Reference URI="#_E379DA17B63A8A4A231615881383049164">
      <ds:Transforms>
        <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
          <ec:InclusiveNamespaces xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#" PrefixList=""/>
        </ds:Transform>
      </ds:Transforms>
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue>Q0v8wzrdvVPIWeFJKqj0anmXwCw=</ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>*==</ds:SignatureValue>
  <ds:KeyInfo Id="KI-E379DA17B63A8A4A231615881383064166">
    <wsse:SecurityTokenReference wsu:Id="STR-E379DA17B63A8A4A231615881383064167">
      <ds:X509Data>
        <ds:X509IssuerSerial>
          <ds:X509IssuerName>CN=VDV Level 2 ION CA 2,O=VDV eTicket Service GmbH & Co.
            KG,C=DE</ds:X509IssuerName>
          <ds:X509SerialNumber>850158900547048</ds:X509SerialNumber>
        </ds:X509IssuerSerial>
      </ds:X509Data>
    </wsse:SecurityTokenReference>
  </ds:KeyInfo>
</ds:Signature>
```

6.3.2.4 SOAP header

Non-encrypted SOAP header that is used to transport routing information in [ionRoutingHeader](#) as well as security information stored in the [Security header](#). The example uses a message to set a as service available in the CRE.

```
<soapenv:Header>
  <!-- Not for synchronous responses and business exceptions-->
  <ionRoutingHeader xmlns="https://eticore.org/ion/communication/3">
    <senderId>6598</senderId>
    <senderRole>1</senderRole>
    <senderService>ccp</senderService>
    <messageNumber>4711</messageNumber>
    <!--Optional, set only if greater 0:-->
    <repeatCounter>1</repeatCounter>
    <processInstanceId>SetServiceAvailable_5f7d9b47-b434-416a-9c60-adfde71dec1e</processInstanceId>
    <receiverId>5602</receiverId>
    <receiverRole>254</receiverRole>
    <receiverService>cre</receiverService>
    <interfaceVersion>3</interfaceVersion>
    <operationName>setServiceAvailable</operationName>
    <!--Optional:-->
    <optionalRoutingInfo>for regional purposes</optionalRoutingInfo>
  </ionRoutingHeader>
  <!-- Security header meeting the requirements of Appendix: etiCORE Standard-Policy -->
  <wsse:Security ...>
    See Security header
  </wsse:Security>
</soapenv:Header>
```


6.3.2.5 Security header

Security header which contains

- Timestamp
- Information about the encryption key used for the [SOAP Body](#)
- Information about the signature of the SOAP body:
 1. Information about canonicalisation and digest method
 2. The signature value
 3. The signature (public) key information

Note: the payload in the SOAP body is signed first with the [sender private signature key](#) and then encrypted with the [Receiver public encryption key](#).

```
<wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
  soap:mustUnderstand="1">
  <wsu:Timestamp wsu:Id="TS-1e340445-f7f9-408a-8c0d-f8f6a4be4be9">
    <wsu:Created>2022-08-22T09:58:16.423Z</wsu:Created>
    <wsu:Expires>2022-08-22T10:03:16.423Z</wsu:Expires>
  </wsu:Timestamp>
  <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" Id="EK-f1763bae-1c1a-45b0-8f5a-70cc927818cb">
    <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p"/>
    <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      <wsse:SecurityTokenReference>
        <ds:X509Data>
          <ds:X509IssuerSerial>
            <ds:X509IssuerName>CN=VDV Level 2 ION CA 2,O=VDV eTicket Service GmbH & Co. KG,C=DE</ds:X509IssuerName>
            <ds:X509SerialNumber>332756131495662</ds:X509SerialNumber>
          </ds:X509IssuerSerial>
        </ds:X509Data>
      </wsse:SecurityTokenReference>
    </ds:KeyInfo>
    <xenc:CipherData>
      <xenc:CipherValue>
        <!-- contains the encrypted symmetric key used for encryption of signature and body -->
      </xenc:CipherValue>
    </xenc:CipherData>
    <xenc:ReferenceList>
      <!-- points to the encrypted body (body not shown in this example) -->
      <xenc:DataReference URI="#ED-2c8f3615-6498-4104-860f-035f1ae21883"/>
      <!-- points to the encrypted signature -->
      <xenc:DataReference URI="#ED-9b6616fd-0851-46d4-9bbb-fab3b45195ce"/>
    </xenc:ReferenceList>
  </xenc:EncryptedKey>
  <xenc:EncryptedData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" Id="ED-9b6616fd-0851-46d4-9bbb-fab3b45195ce"
    Type="http://www.w3.org/2001/04/xmlenc#Element">
    <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes256-cbc"/>
    <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      <wsse:SecurityTokenReference
        xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
        xmlns:wsse11="http://docs.oasis-open.org/wss/oasis-wss-wssecurity-secext-1.1.xsd"
        wsse11:TokenType="http://docs.oasis-open.org/wss/oasis-wss-soap-message-security-1.1#EncryptedKey">
          <wsse:Reference URI="#EK-f1763bae-1c1a-45b0-8f5a-70cc927818cb"/>
        </wsse:SecurityTokenReference>
      </ds:KeyInfo>
      <xenc:CipherData>
        <xenc:CipherValue>
          <!--
            Contains the encrypted signature and its metadata. The decrypted content would look like this:
            see Signature
          -->
        </xenc:CipherValue>
      </xenc:CipherData>
    </xenc:EncryptedData>
  </xenc:EncryptedData>
</wsse:Security>
</soap:Body>
</soap:Envelope>
```

```
-->
</xenc:CipherValue>
</xenc:CipherData>
</xenc:EncryptedData>
</wsse:Security>
```

6.3.2.6 Encrypted payload

Helper element that indicates that the message payload is not embedded directly in the [SOAP Body](#) but first encrypted.

6.3.2.7 Request payload type

This class represents the conceptual payload part of a request message as a data type, instantiated by [request payload](#).

As a data structure, it is a placeholder for all business request data sent via the ION.

The request payload consists of the [CommunicationInformation](#) and optionally contains an additional [Request business data](#) part.

6.3.2.8 Request business data

This element serves as a conceptual placeholder for further business data to be determined by the [Initiator](#), which is to be sent together with the general parts of a request and is needed on the [Processor](#) side to process the request.

Depending on the use case, the name of the request and its general data may be sufficient to process the request (e.g. when retrieving a hotlist for entitlements).

In this case, this element is not included.

6.3.2.9 Reponse payload type

This class represents the conceptual payload part of a response message as a data type, instantiated by [response payload](#).

As a data structure, it is a placeholder for all business responses data sent via the ION.

The Response payload type is always a [BusinessAcknowledgement](#) with a [CommunicationInformation](#) object, an [OriginalRequestCommunicationInformation](#) object and an optional (depending on the use case) [Response business data](#) object.

6.3.2.10 Response business data

This conceptual element contains further business data to be processed by the [Initiator](#).

A response business data occurs if the use case delivers data beyond a [BusinessAcknowledgement](#). This is, for example, the case in get-scenarios. The hotlist service system taken as an example, the response business data would be the hotlist itself.

6.3.2.11 Signature key reference

Reference to the signature key. It is part of the [Signature](#) Information and located in the [Security header](#).

See also [Sender private signature key](#) and [Sender public signature key](#).

To verify the signature of the message, the [Sender public signature key](#) has to be fetched by this reference from the directory service (LDAP).

6.3.2.12 current message

Current message with [SOAP Envelope](#).

6.3.2.13 message header

Concrete header of a current message embedded in the [SOAP Header](#).

The communication header is unencrypted because routing engines must read this header. The signature of the header is not needed due to transport encryption with client authentication. The communication header contains as a copy the parts from the communication information layer located in [CommunicationInformation](#) that are needed for routing, as well as further information about the service and the interface version. See also [ionRoutingHeader](#).

6.3.2.14 ION message number

Generator or similar which provides or generates a new unique message number for this organisation, role and service. An instance of [IonMessageNumber](#).

6.3.2.15 communication information

An object which is part of the incoming [request payload](#) or, furthermore, of the incoming [response payload](#) in an asynchronous context. Instance of [CommunicationInformation](#).

6.3.2.16 original request communication information

See also type description in [OriginalRequestCommunicationInformation](#).

Contains correlation information to the previous request which caused this response.

Needed additionally to the [CommunicationInformation](#) since these information are needed in asynchronous use cases to transport sender and receiver information in the related response and to build the routing header.

6.3.2.17 request payload

An object which stores the current [request payload](#) for caching reasons.

6.3.2.18 response payload

An instance of the [response payload](#).

6.3.2.19 exception payload

An instance of the [exception payload](#).

6.3.3 Configurations

6.3.3.1 Policy

The class that contains the most important parts of the WSS policy for comprehension purposes. For more information, see chapter [Message-based encryption \(application layer\)](#) where all parts used in etiCORE are explained in detail. For an example see [Appendix: etiCORE Standard-Policy](#)

6.3.3.1.1 AsymmetricBinding

Describes a binding based on asymmetric private/public key procedure.

```
<sp:AsymmetricBinding>
  <wsp:Policy>
    <!-- Client X.509-Certificate -->
    <sp:InitiatorToken>
      <wsp:Policy>
        <!--do NOT include the X509 certificate data in the request,provide issuer reference only -->
        <sp:X509Token sp:IncludeToken=
          "http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/Never">
          <wsp:Policy>
            <sp:WssX509V3Token10/>
            <sp:RequireIssuerSerialReference/>
          </wsp:Policy>
        </sp:X509Token>
      </wsp:Policy>
    </sp:InitiatorToken>
    <!-- Server X.509-Certificate -->
    <sp:RecipientToken>
      <wsp:Policy>
        <!-- do NOT include the X509 certificate data in the request,
          provide issuer reference only -->
        <sp:X509Token sp:IncludeToken=
          "http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/Never">
          <wsp:Policy>
            <sp:WssX509V3Token10/>
            <sp:RequireIssuerSerialReference/>
          </wsp:Policy>
        </sp:X509Token>
      </wsp:Policy>
    </sp:RecipientToken>
    <sp:Layout>
      <wsp:Policy>
        <!-- Use strict Header Layout -->
        <sp:Strict/>
      </wsp:Policy>
    </sp:Layout>
    <!-- Provide creation date and expiry in the request -->
    <!-- Note: Standard expiry of 5 minutes must be increased to 7 days
      for asynchronous requests -->
    <sp:IncludeTimestamp/>
    <sp:OnlySignEntireHeadersAndBody/>
    <sp:AlgorithmSuite>
      <wsp:Policy>
        <sp:Basic256Sha256/>
      </wsp:Policy>
    </sp:AlgorithmSuite>
    <sp:EncryptSignature/>
  </wsp:Policy>
</sp:AsymmetricBinding>
```

6.3.3.1.2 EncryptedParts

Defines the parts in the message to be encrypted. See [Encryption Scope](#).

```
<sp:EncryptedParts>
  <sp:Body/>
</sp:EncryptedParts>
```

6.3.3.1.3 SignedParts

Defines the parts in the message to be signed. See [Signature Scope](#).

```
<sp:SignedParts>
  <sp:Body/>
</sp:SignedParts>
```

6.3.3.2 webservice security policy

An instance of the currently used WSS policy. See also chapter [Webservice Security](#).

6.3.3.3 current service

Current service inside the role. Usually configured.

6.3.3.4 current organisation ID

Current organisation ID. Usually configured.

6.3.3.5 current role

Current role. Usually configured.

7 Non-Functional Requirements

This chapter lists all non-functional requirements for the etiCORE processes where the ION is involved.

7.1 Process Instance ID Handling

This chapter describes the creation and handling of [ProcessInstanceId](#) fields in messages.

The system component that is responsible for creating a [ProcessInstanceId](#) depends on the start location of the process.

If the start point is a terminal, the [ProcessInstanceId](#) is embedded in the [TCommunicationInformation](#) of a t-message (message from a terminal to a back-office system).

If the start point is a back-office system, the [ProcessInstanceId](#) is embedded in the [IonRoutingHeader](#) and in [CommunicationInformation](#) of an ION message.

7.1.1 Handle process instance ID

Activity to handle a [ProcessInstanceId](#). Either a new one is generated or an existing one is retained.

A [ProcessInstanceId](#) contains the name of the basic process and a UUID. To generate a process instance ID, take the basic process name from [ProcessNameEnum](#) and a UUID (in canonical string form) and combine them separated by an underscore:

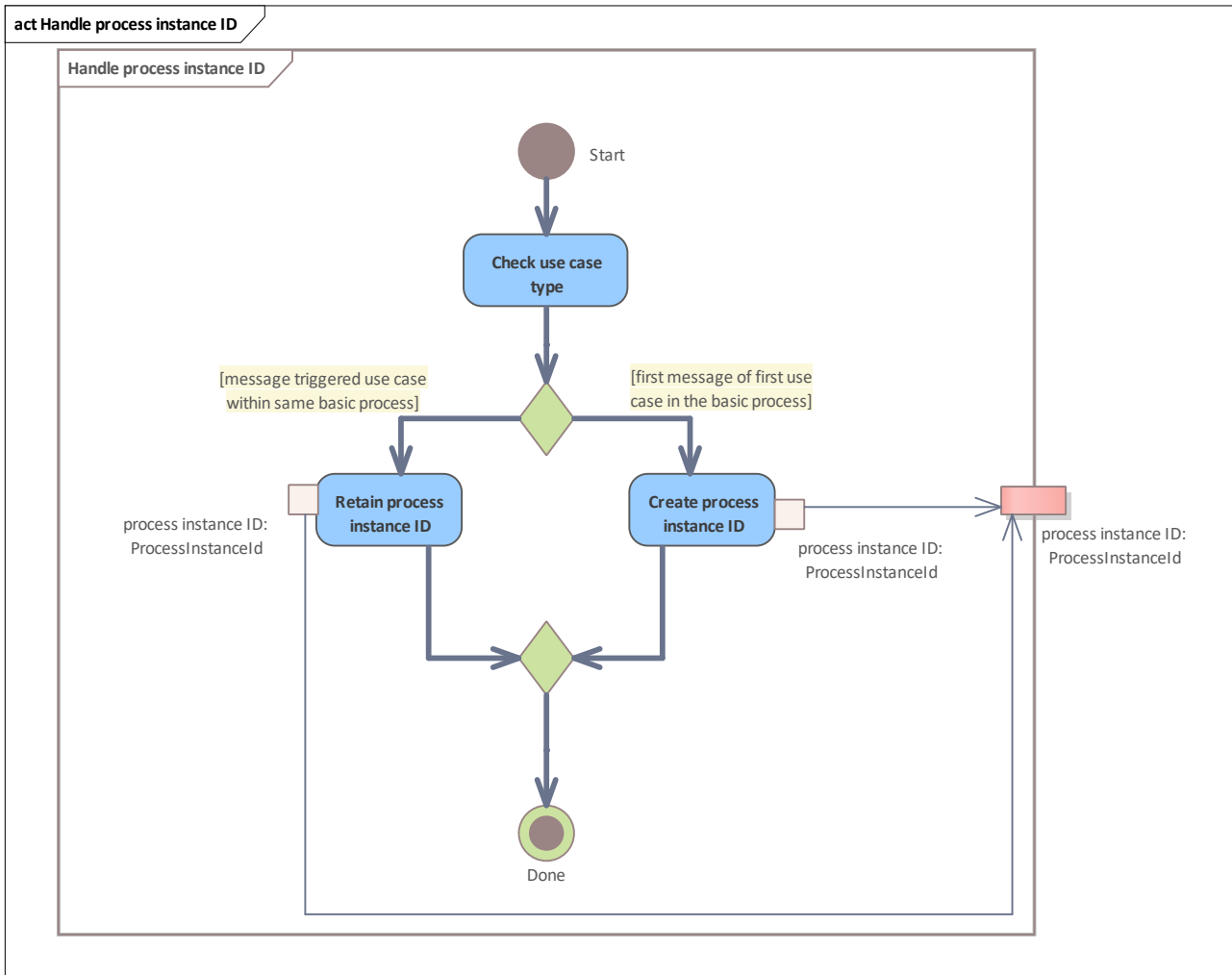
<process name>"_"<UUID>

Example:

IssueEntitlement_00000000-0000-0000-0000-000000000000

In this example, the process starts in a CCP terminal, thus the [ProcessInstanceId](#) has to be created there.

7.1.1.1 Handle process instance ID



See [Initiator::Handle process instance ID](#).

Check use case type

Checks if the underlying use case is responsible for the first message in a starting basic process or if an existing [ProcessInstanceId](#) (e.g. coming from a terminal message) has to be retained.

Create process instance ID

If the underlying use case is the first one in a processing or notification chain, a new [ProcessInstanceId](#) has to be created. This [ProcessInstanceId](#) will be used in the whole message chain.

Retain process instance ID

Retain an existing [ProcessInstanceId](#) coming from a terminal or other ION system in the same basic process.

All communication parts of a certain underlying basic process can be found in the basic processes defined in [Layer 2 - Basic Processes as BPMN Collaboration](#). In the BPMN collaboration diagrams, the communication of a basic process between the participants is shown. During this basic process the same [ProcessInstanceId](#) is used.

7.2 Message Timeout and Retry Handling

Due to certain error scenarios, message retry becomes important.

Overall rules:

1. if a [SOAP body](#) has to be changed except for message timestamp and repeat counter (in [CommunicationInformation](#)) a new [IonMessageId](#) has to be used. If the [ionRoutingHeader](#) has to be changed, except for the repeat counter and optional routing info, a new [IonMessageId](#) has to be used.
2. If a [BusinessException](#) occurs, a new [IonMessageId](#) has to be used (except for the exception [E ION_DUPLICATE_ION_MESSAGE_ID](#))

Note: the repeat counter is part of the body and must match the one in the [ionRoutingHeader](#).

We distinguish three different scenarios for a potential message retry:

- Send the same message again, including the same [IonMessageId](#) if it is sure that the message never reached the intended recipient. The repeat counter can optionally be incremented by one
- Send the same message again, including the same [IonMessageId](#) if it is not sure that the message reached the intended recipient. The repeat counter can optionally be incremented by one. In case of a [E ION_DUPLICATE_ION_MESSAGE_ID](#), the same message (same [IonMessageId](#)) with an incremented repeat counter and a new message timestamp can be sent.
- Send the (possibly corrected) content of the message again with a new [IonMessageId](#)

In the first case, certain communication error allows an unchanged message (except for message timestamp and repeat counter) to be resent. These errors are

- [E IONC_RECEIVER_NOT_REACHABLE](#)
- [E IONC_UNKNOWN_TECHNICAL_PROBLEM](#)
- [E IONC_CRE_COULD_NOT_STORE_MESSAGE](#)
- [E IONC_NO_TARGET_ADDRESS_FOUND_FOR_ROUTING_PARAMETERS](#) (after fixing the configuration of the CRE, if possible)
- [E IONC_UNKNOWN_OPERATION](#) (after fixing the configuration of the CRE if possible)
- [E IONC_UNKNOWN_OR_UNAUTHORISED_RECEIVER](#) (after fixing the configuration of the CRE, if possible)
- [E IONC_UNKNOWN_OR_UNAUTHORISED_SENDER](#) (after fixing the configuration of the CRE, if possible)

In the second case, the error [E IONC_RECEIVER_RESPONSE_TIMEOUT](#) might occur or the reply could not be received within the configured timeout interval for asynchronous replies. Then, the [Initiator](#) does not know the processing state. To prevent the [Processor](#) system from processing the message again (if it already processed it), the same [IonMessageId](#) plus an increased repeat counter has to be used. These are the timeout scenarios which are described in detail in [Timeout Handling](#).

In the third case, a set of communication errors may occur which indicates a misconfiguration or message inconsistency to be fixed before sending the message again with a new [IonMessageId](#):

- [E IONC_HEADER_TO_TLS_CERTIFICATE_MISMATCH](#)
- [E IONC_INVALID_SOAP_HEADER](#)
- [E IONC_NO_TARGET_ADDRESS_FOUND_FOR_ROUTING_PARAMETERS](#) (after fixing the routing parameters in the message)
- [E IONC_UNKNOWN_OPERATION](#) (after fixing the routing parameters in the message)

- [E IONC UNKNOWN OR UNAUTHORISED RECEIVER](#) (after fixing the routing parameters in the message)
- [E IONC UNKNOWN OR UNAUTHORISED SENDER](#) (after fixing the routing parameters in the message)

Furthermore, if the CRE could deliver the message, a business exception may have been sent to indicate that the message content caused processing problems.

A business exception means that the [Processor](#) system did not process the message (except for logging purposes). This allows a corrected message with a new [IonMessageId](#) to be sent.

7.2.1 Timeout Handling

The ION timeout handling distinguishes three different scenarios:

- A direct network timeout due to exceeding a configured timeout interval for a certain operation
- A message timeout caused by clock misconfiguration or a non-sufficient WSS timestamp expiry date
- A message timeout in asynchronous context. The reply was not received within the SLA interval. Note: such a timeout can also be caused due to caching the message in the CRE (see also [\[3\] CRE Specification: Message Caching](#))

Furthermore, [Processor Behaviour](#) describes the [Processor](#) side.

7.2.1.1 Direct Timeout

Since the CRE is placed "in the middle" of the network route, a direct timeout may occur between the [Initiator](#) and the CRE or the CRE and the [Processor](#). The former problem is rather simple to fix. Since the message has not been processed it can be repeated unchanged once the network problems between Initiator and CRE have been fixed.

The more complex timeout occurs if the [Initiator](#) sends a message via the CRE to the [Processor](#) and the configured time interval in the CRE for the duration of the operation is not sufficient to get the reply from the [Processor](#).

- The CRE closes the connection and sends an [E IONC RECEIVER RESPONSE TIMEOUT](#) to the [Initiator](#)
- The [Initiator](#) must react. To indicate that it could not get the reply, It must resend the message (same [IonMessageId](#)) with incremented repeat counter at a later time with a new message timestamp. Caution: the WSS expiration timestamp also has to be changed. This is normally done automatically by the WSS framework

Mitigation

- The timeout interval in the CRE can be increased. This is only allowed if the problem occurs with several recipients
- If only one recipient is involved, the recipient has to improve its data processing to prevent these problems in the future
- If the use case covers a [Get Scenario](#), the scheduled time of the [Initiator](#) can be changed. Maybe the current time is used by too many [Initiators](#)

Note: The timeout configuration per operation is done in the CRE. This configuration does not cover the overall timeout interval, which has to be configured on initiator side. This interval has to consider

- Network latency between client and CRE
- The configured interval in the CRE

- Network latency between CRE and client

For timeout configuration on the initiator side, we recommend double the interval which is configured in the CRE. To enable this, see [Appendix: CRE Timeout Configuration Table](#).

7.2.1.2 WSS Timestamp Expiration

The WSS timestamp expiration period has to be set to 7 days. A typical default value is only 5 minutes. If a system does not set this value, the too-short default might cause a problem.

Error scenario

- The client gets a runtime exception which indicates that the message was discarded due to the WSS timestamp check reason

Mitigation

- Check that the client system sets the expiration value correctly to 7 days
- Check that the server system accepts messages with the expiration value of 7 days
- Fix the problem and resend the message with a modified message timestamp, modified WSS expiration timestamp and same [IonMessageId](#). Optionally, the repeat counter can be incremented by one.

7.2.1.3 SLA Timeout

SLA timeout means that the message but could not be sent to the intended recipient within the maximum message expiry interval described in [\[5\] SLAs for Message Transfer](#). This can only happen in asynchronous context.

Either the message of the [Initiator](#) never reached the intended [Processor](#) or the [Initiator](#) never got the reply of the [Processor](#).

The message(s) without reply have to be identified and can then be sent with unchanged business data. To ensure receiving the original reply (if it exists), the messages must use their original [IonMessageId](#) with a new timestamp in [CommunicationInformation](#) and an incremented repeat counter in [CommunicationInformation](#) and [ionRoutingHeader](#).

7.2.1.4 Processor Behaviour

If a message reaches the [Processor](#), it must distinguish different cases. In asynchronous context, the [deliveryAcknowledgement](#) is sent before the [IonMessageId](#) is registered, therefore the behaviour is the same for synchronous or asynchronous context:

- The basic path: the message is processed and the reply is sent ([IonMessageId](#) unknown, repeat counter not set or arbitrary), send [Reply](#) without repeat counter
- A message with the tuple [IonMessageId](#) and repeat counter has already been processed: throw [E ION_DUPLICATE_ION_MESSAGE_ID](#)
- A message with the [IonMessageId](#) is known and already processed, the repeat counter of the incoming message is set and arbitrary (> 0 , current counter value not yet in system): Do not process the message again. Register [IonMessageId](#) with the current repeat counter. Fetch saved reply message (not considering exception messages containing [E ION_DUPLICATE_ION_MESSAGE_ID](#) -> there must be exactly ONE left, which might still be an exception message). Send the [Reply](#) with an incremented repeat counter and leave the originalRepeatCounter as it is.
- A message with the [IonMessageId](#) is known and still being processed, the repeat counter of the incoming message is set and arbitrary (> 0 , current counter value not yet in system): Register [IonMessageId](#) with the current repeat counter. Drop the current message. Send the

[Reply](#) when the previous incoming message with the same [IonMessageId](#) has been processed with the repeat counter (if any) of the previous message.

7.2.1.5 Appendix: CRE Timeout Configuration Table

Table to configure network timeout for certain operations.

Operation	Service	Role	Time out/s
getApplicationHotlist	hotlist	5	60
getEntitlementHotlist	hotlist	5	60
getEntitlementWithProductHotlist	hotlist	5	60
getOrganisationHotlist	hotlist	5	30
getAuthencationKeyHotlist	hotlist	5	30
getSamHotlist	hotlist	5	30
getIncrementalApplicationHotlist	hotlist	5	30
getIncrementalEntitlementHotlist	hotlist	5	30
getUnclaimedListInformation	hotlist	5	30
verifyApplicationHotlistUpdatedViaIncrements	hotlist	5	30
verifyEntitlementHotlistUpdatedViaIncrements	hotlist	5	30
all further operations	hotlist	5	5
processCheckinNotificationList	po	3	60
processCheckoutNotificationList	po	3	60
processEntitlementInspectedNotificationList	po	3	60
processEntitlementIssuedNotificationList	po	3	60
all further operations	po	3	5
getActionList	po-oa-management	3	60
getIncrementalActionList	po-oa-management	3	30
verifyActionListUpdatedViaIncrements	po-oa-management	3	30
all further operations	po-oa-management	3	5
processStaticEntitlementInspectedNotificationList	po-ste	3	60
processStaticEntitlementIssuedNotificationList	po-ste	3	60
all further operations	po-ste	3	5
all operations	ccp	1	5
all operations	ccp-oa-ordering	1	5
all operations	ccp-oa-execution	1	5
all operations	so	2	5
all operations	so-ste	2	5
all operations	esh	4	5
all operations	cre	254	5

7.2.2 Retry Handling

This chapter contains retry scenarios for direct timeout and SLA timeout.

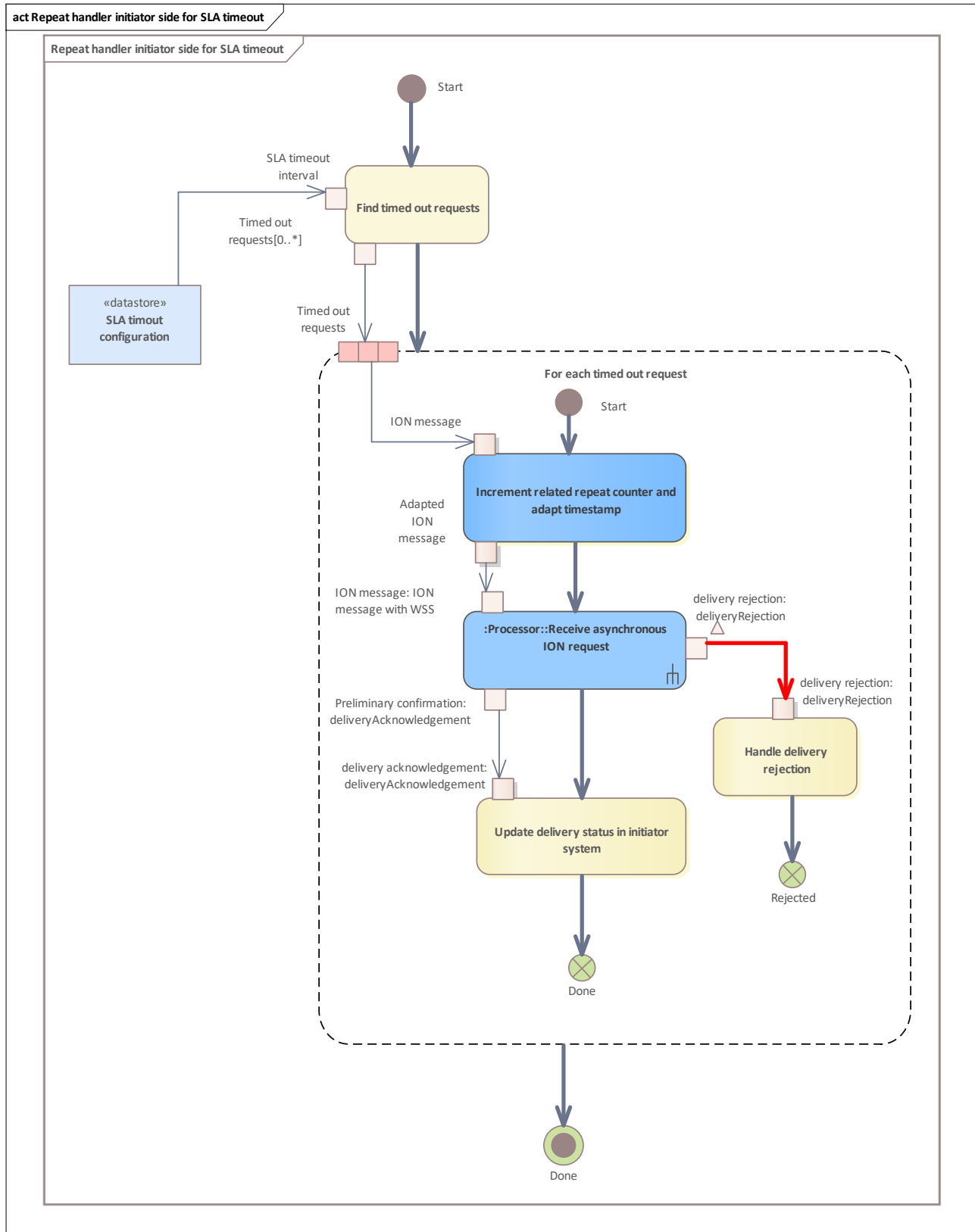
Repeat handler sender side for direct timeout

See [Repeat handler sender side for direct timeout](#).

7.2.2.1 Repeat handler initiator side for SLA timeout

A batch running process has to determine whether an SLA timeout occurred on any sent messages in the system.

7.2.2.1.1 Repeat handler initiator side for SLA timeout



Find timed out requests

Iterate over the database via batch process or similar and find all requests where a [deliveryAcknowledgement](#) exists but no asynchronous reply was referred to within the SLA time span.

Increment related repeat counter and adapt timestamp

If the repeat counter was not set yet, set it to 1, otherwise increment it by one. Adapt the timestamp to "now".

Processor::Receive asynchronous ION request

See [Processor::Receive asynchronous ION request](#).

Handle delivery rejection

Action that handles a potential [deliveryRejection](#). This reply has to be evaluated and then, the right actions must be taken depending on the use case.

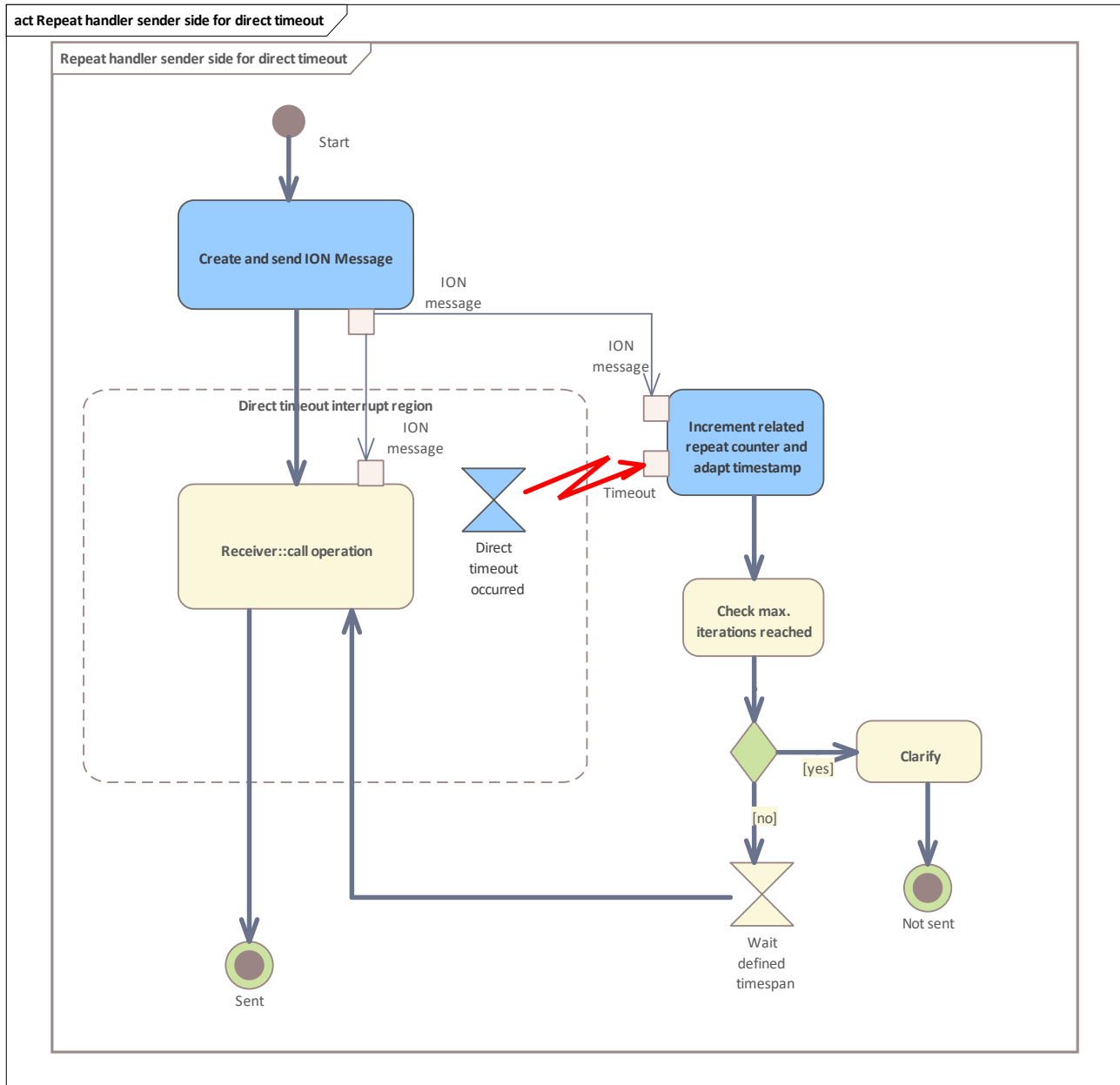
Update delivery status in initiator system

Register [deliveryAcknowledgement](#) as preliminary confirmation, so that the request can be processed in the [Processor](#)'s business logic. The next step is to wait the SLA timespan for the final business response (see [\[5\] SLAs for Message Transfer](#)).

7.2.2.2 Repeat handler sender side for direct timeout

Handle a retry scenario on sender side for a direct timeout.

7.2.2.2.1 Repeat handler sender side for direct timeout



Receiver::call operation

Placeholder for an arbitrary operation call.

Increment related repeat counter and adapt timestamp

If the repeat counter was not set yet, set it to 1, otherwise increment it by one. Adapt the timestamp to "now".

Wait defined timespan

Wait a defined time span before a retry is performed. The time span may vary due to the current number of iterations.

Check max. iterations reached

Try a certain number of iterations to resend the message automatically. If the maximum amount is reached, the automatic retry process stops. A manual clarification process can then be started.

Clarify

The message could not be delivered.

This requires manual steps to find out, why the other party is not accepting the message.

Create and send ION Message

Create and send ION message like described in the use case [Compile and send ION request](#).

Direct timeout occurred

A timeout occurred. The timeout interval depends on the kind of operation and can be found in [Appendix: CRE Timeout Configuration Table](#).

7.3 Version Management

A core function of (((etiCORE is the handling of different versions.

Version always means the first number within a three-parts release:

[Version].[Service Pack].[Build]. The version itself is mapped in the namespace of the interfaces, which thus also becomes version-safe.

Version management is not mapped in separate fields within the message but exclusively via the namespace.

Furthermore, the versions are separated by URLs:

- The CRE provides one URL per version.
- Each participant system provides one URL per version for its services.

In the diagrams in the following chapters, it can be seen how version management is mapped declaratively from the outside via different URLs without having any influence on the messages themselves.

In order to achieve this, the version is recorded in the SOAP header and is necessary as routing information. The version is stored in the [ionRoutingHeader](#) of the header.

Based on the routing information, the CRE determines the address of the service to be called.

The version information is included here.

It must be possible to support two versions simultaneously.

If the transported message information is sufficiently similar in the different versions, the versions can merge again in the database of the end system at the latest.

The two versions mentioned are the previous version KA 1.X with the namespaces 1 and 2 in the interfaces and the current version of (((etiCORE with the namespace 3 in the interfaces.

7.3.1 One Version per Participant

This diagram shows the version handling in the easy case, that one system with one version communicates with the CRE and the other system with the same version - and only one version is shown per system. Since two versions have to be supported, this diagram only shows the principle.

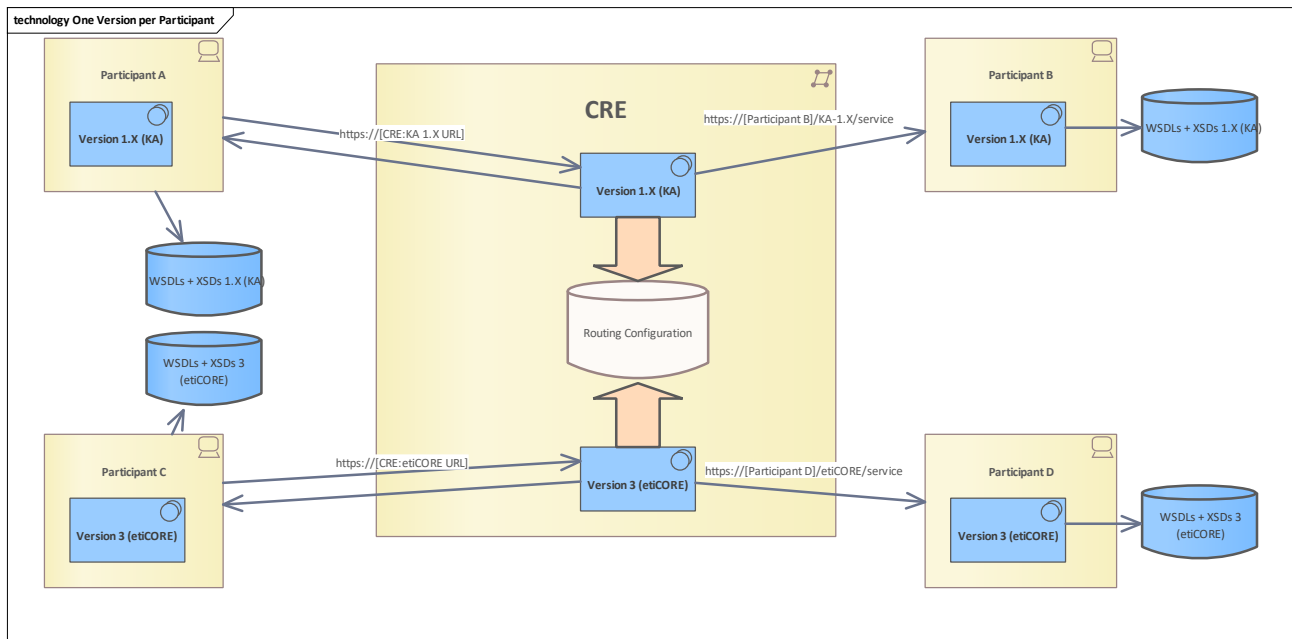


Figure 33: One Version per Participant

7.3.2 One Participant serves two Versions

This diagram shows the version handling in the more complex case, that one system with one version communicates with the CRE and the other system with the same version - but Participant E is able to serve both versions. Since two versions have to be supported by each system, this diagram only shows the principle, that the system (Participant E) has to respond in the same version it was requested.

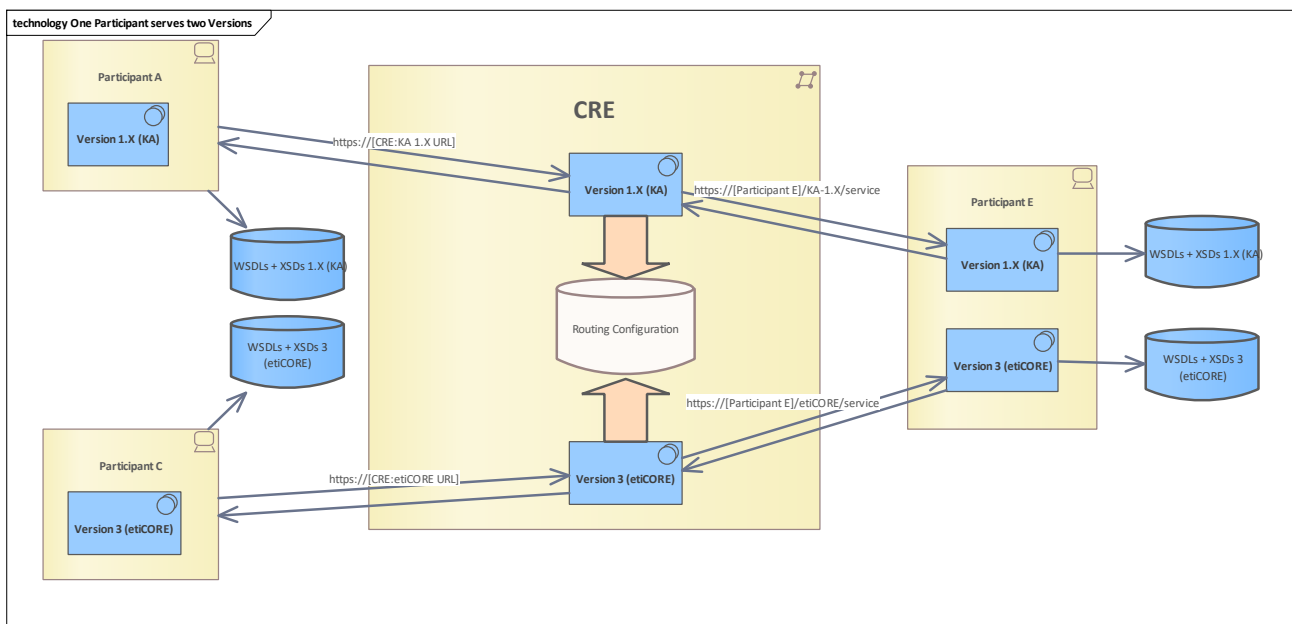


Figure 34: One Participant serves two Versions

7.3.3 Each Participant serves two Versions

This diagram shows the version handling in the most complex case, that each system is able to communicate with the CRE in both versions. Since two versions have to be supported by each system, this diagram only shows the truth if nationwide interoperability is employed.

Note: as long as old versioned user media and SAMs are used, both versions are needed.

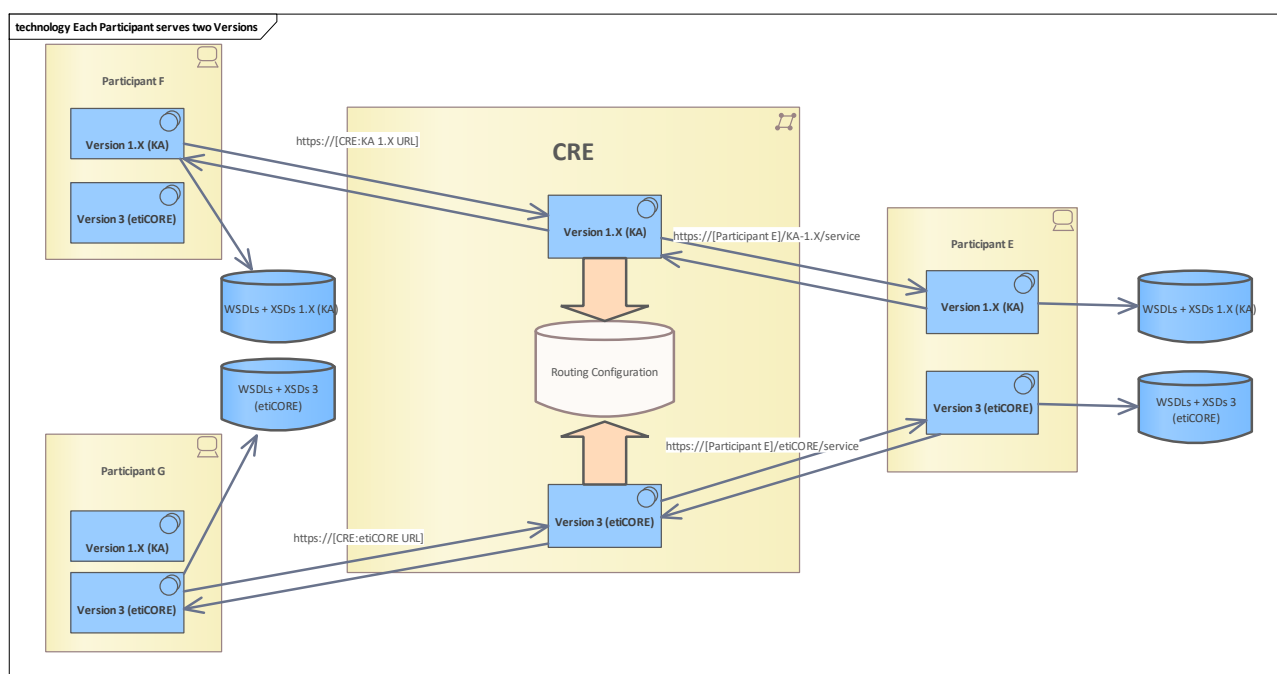


Figure 35: Each Participant serves two Versions

7.4 Common Non-Functional Requirements

This chapter lists all non-functional requirements for the etiCORE processes where the ION is involved.

7.4.1 Messaging requirements

Messaging requirements that have to be fulfilled in order to enable the data exchange in the basic processes of the etiCORE standard.

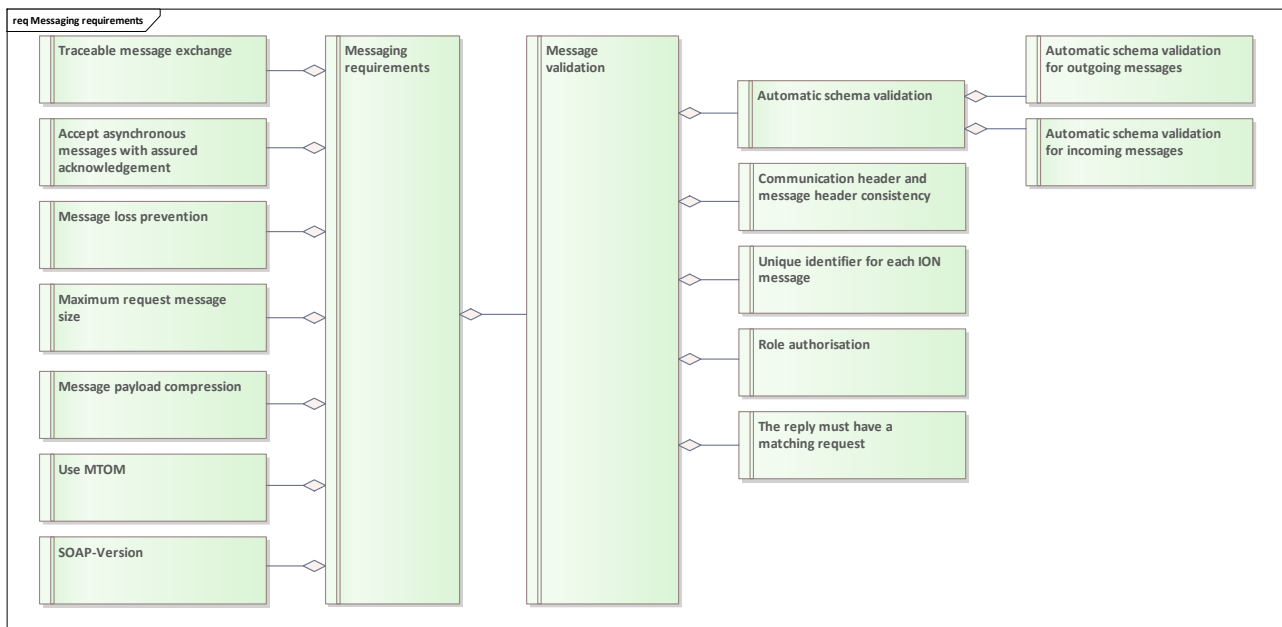


Figure 36: Messaging requirements

Diagram for [Messaging requirements](#).

7.4.1.1 Security requirements

This chapter describes the important ION requirements concerning security inside the (((etiCORE.

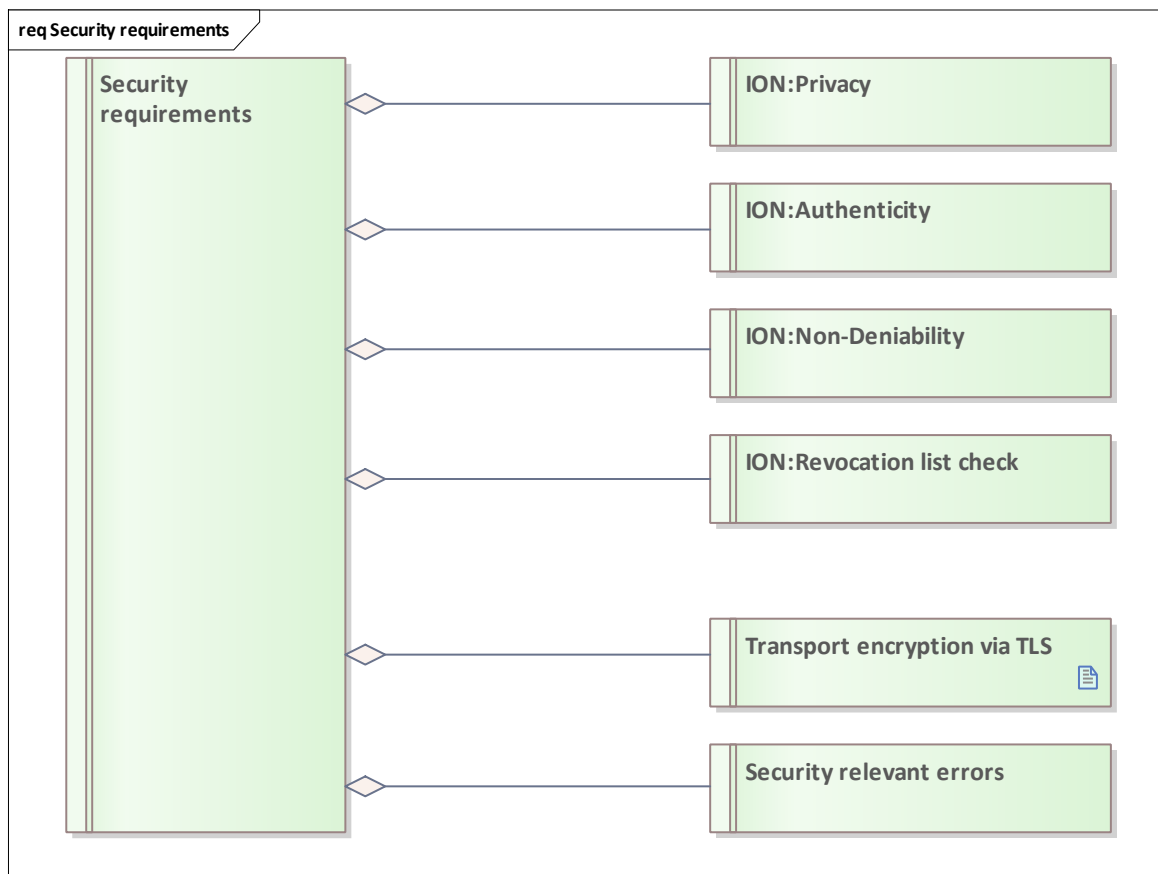


Figure 37: Security requirements

This diagram shows the requirements concerning security when exchanging ION messages.

7.4.1.1.1 Security requirements

Root requirement for the security inside the ION (interoperable network). Consists of the requirements

- [ION:Non-Deniability](#)
- [ION:Authenticity](#)
- [ION:Privacy](#)
- [ION:Revocation List Check](#)
- [Security relevant errors](#)
- [Transport encryption via TLS](#)

The transmission of messages via the ION considers the data protection law requirements. The essential criteria for this are the following points:

- Personal data is transferred that could be used to identify a customer.
- Identifiers are used that could be used in the CCP systems to assign movement data and customers (pseudonyms).

7.4.1.1.2 ION:Privacy

This requirement concerns the ION communication. If a sender sends a message, it has to be sure that no one else but the receiver is able to read this message.

The ION communication must ensure this privacy.

This is done by using XML encryption in the messages. The sender encrypts the message with the (certificated) public key of the receiver. The receiver is the only instance that has the private key for decryption of the message.

7.4.1.1.3 ION:Authenticity

This requirement concerns the ION communication. If a sender sends a message to the receiver, the receiver must be sure that the message was not changed by a third party and that the message was sent by the sender which is referenced in the message.

This is done by using an XML signature in the messages. Due to the signature (and the verification of the signature), this requirement is fulfilled, as the signature verification fails if the message is corrupted.

7.4.1.1.4 ION:Non-Deniability

This requirement concerns the ION communication. If a sender sends a message and the message was acknowledged technically by the receiver, the sender can be sure that its message receipt will not be denied afterwards.

The ION communication must ensure this non-deniability.

This is done by using an XML signature in the messages. During synchronous communication, the direct response is signed by the receiver and verified by the sender. In this case, non-deniability is achieved directly.

During asynchronous communication, the short acknowledgement [deliveryAcknowledgement](#) is not sufficient for the sender. Mutual non-deniability is only achieved once the receiver responds with a processed reply message, either a regular response or a defined business exception.

7.4.1.1.5 ION:Revocation list check

This requirement forces a frequent check of certificates against the revocation lists coming from the PKI. This covers the check of each employed leaf certificate as well as the related Sub-CA and Root-CA certificates.

For details, see also [Distribution of the Certificates](#).

7.4.1.1.6 Transport encryption via TLS

For transport encryption (TLS for HTTPS), each organisation uses an up-to-date key pair consisting of a public key certified by X.509v3 certificate with the corresponding private key. The transport security takes place between the sender and the CRE on the one hand and between the CRE and the receiver on the other hand.

TLS >= version 1.3 is used in the context of transport security (see [\[12\] RFC 5246](#)). Using HTTP as protocol, HTTPS is used together with TLS.

Both client and server authentication take place. The client proves its authenticity to the server with its certificate. The server in turn proves its authenticity to the client.

If a new HTTPS connection must be established to send a message, check that the CRE presents a valid certificate that is trusted and issued to the CRE by a valid (Sub-) Root CA.

For clients without a valid certificate, the connection to the server is refused, thus eliminating the need for any other authentication (e.g. basic authentication via username and password) in order to send messages.

The certificate (client or server) for authentication within the TLS handshake is bound to the organisation ID of the company operating the system, but not to its role.

By using a central (and possibly several local routers of a JSB), the protection of the message content cannot be guaranteed by the transport encryption via TLS alone.

For this reason, the requirements [ION:Privacy](#) and [ION:Authenticity](#) applied directly to the message must also be fulfilled by employing WSS. See [Webservice Security](#).

During the TLS handshake, the central routing engine (CRE) checks with the client whether the organisation ID used in the TLS certificate matches the sender ID in the [IonRoutingHeader](#).

The check must take place in the CRE, as the final recipient cannot make a check in this regard:

- the SOAP header is not signed
- the certificate of the CRE is used for transport security to the final recipient and not the certificate of the sender.

Since the JSB establishes the TLS connection to the CRE, the organisation ID of the JSB and not that of the sender is then in the TLS certificate. Therefore, the JSB must be authorised at the ZVM. This means that certain organisations may establish a TLS connection using the JSB without the CRE rejecting the message due to a mismatch of the organisation ID in the TLS certificate and the sender ID in the [IONRoutingHeader](#).

To do this, the JSB is registered and configured in the registrar (ESH). The data is then propagated to the CRE.

On the other hand, in ION, the receiving system (participant or JSB) has to check that the TLS certificate is owned by the CRE, since the CRE is the only instance that is allowed to communicate with either the participant system or the JSB.

TLS Attributes

Protocol	TLS Version 1.3 or higher
Client Authentication	Yes
Server Authentication	Yes
Cipher Suite	Negotiated during TLS handshake

Certificates

Standard	X.509
Version	V3
Key Algorithm	RSASSA-PSS
Key Length	2048 Bits

7.4.1.1.7 Security relevant errors

All security-relevant error cases point to a misuse of the system. Likewise, a systematic attack or even an attempted DoS (Denial of Service) attack could be the cause.

For this reason, the system must never provide internal information that indicates security problems; in particular, no responses with corresponding codes may be generated.

Especially for DoS attacks, a behaviour must be chosen that consumes as few resources as possible (usually only log entry, possibly alarm and HTTP error code).

This applies to

- Messages that cannot be decrypted or are not encrypted
- Unsigned or incorrectly signed messages
- Messages that do not comply with the XSD

7.4.1.2 Messaging requirements

Parent requirement for all requirements which deal with messaging. Has to be fulfilled or respected by all use cases which employ ION messaging.

7.4.1.3 Traceable message exchange

The [ProcessInstanceId](#) offers the possibility to uniquely identify the distributed message exchange based on a process. In doing so, a unique ID with process name and UUID is to be assigned at the process initiator, which is retained for the entire message exchange. In this way, a JSB or also the CRE, can assign messages to a process. The process instance ID is transmitted both in the SOAP header and in the message payload. See [Initiator::Handle process instance ID](#) for more details.

Note: User-medium-based processes usually start in the terminal. In a subsequent message exchange to the back-office systems, the process instance ID must be assigned in the terminal, transmitted in the message to the back-office system and then used later in the back-office system for messages of the same process.

7.4.1.4 Message loss prevention

If the CRE replies to the message sender with a [deliveryAcknowledgement](#), the responsibility of the message transmission is transferred to the CRE.

This applies in the case of intermediate message storage in the CRE.

The CRE uses a mechanism to deliver messages repeatedly when the recipient is unavailable. If the services of a recipient are not available, the CRE temporarily stores the messages in a queue.

If the unavailability is caused because the recipient has logged out of the CRE, the next delivery attempt of the queued messages will only take place after the service provider has logged in again.

In the other case - the service provider is unexpectedly unavailable and has not logged off beforehand - an attempt is made to deliver the messages to the service provider at preconfigured intervals.

With each unsuccessful redelivery attempt, the period of time until a new delivery is attempted increases. If the message could not be delivered within 7 days, the message is placed in a dead letter queue and the original sender of the message is informed that the recipient did not receive the message by [Notify about discarded messages](#).

The detailed store and forward process of the CRE is described in [Process asynchronous message](#), [Store ION message](#), [Deliver queued messages to service](#), [Deliver queued messages in scheduler](#), [Discard queued messages](#) and [\[3\] CRE Specification: Message Store and Forward](#).

7.4.1.5 Maximum request message size

The maximum size of request messages which have to be potentially queued must not exceed 16 MB.

7.4.1.6 Message payload compression

The ZIP standard is used for compression. The participating systems are responsible for this. From the ION point of view, the compression of message parts is transparent, as this is done within the SOAP body.

Compression is not used in all use cases. Where compression occurs, this requirement is linked. These are, for example

- The hotlists and configuration lists contained in the responses from the hotlist service.
- The action lists contained in the responses from Action List Management
- The lists generated in the submission of summarised output, control and capture messages
- The list of organisations and roles to be collected from the Registrar

These XML-format contents are packed via ZIP and then placed in a message tag as Base64 encoding.

The signature is made including this Base64 encoded data in the tag, so that falsification is not possible.

This means that the data including the list is also encrypted within the technical message with the help of the WSS mechanisms used.

On the initiator side, in addition to decryption and signature verification, there is the task of performing Base64 decoding on the tag of the list.

- Base64 decoding on the tag of the list
- and then unpack the remaining binary (zipped) data.

The MIME type indicates that the format is a zipped format. For all lists in the response, there are the fixed data types [CompressedXmlList](#) and [CompressedBusinessReplyList](#). The lists of the summarised notifications vary in the different requests depending on the single notification type.

7.4.1.7 Accept asynchronous messages with assured acknowledgement

This requirement aims at the system behaviour of all participants concerning the processing of asynchronous messages.

The ION interfaces provide asynchronous data flow by employing two synchronous message flows:

1. [Initiator](#) > [ION request message](#) > [Processor](#). [Processor](#) > [DeliveryAcknowledgement](#) > [Initiator](#)
2. [Processor](#) (now sender) > [ION reply message](#) > [Initiator](#) (now receiver). [Initiator](#) > [DeliveryAcknowledgement](#) > [Processor](#)
3. **No further messages**

In order to always determine the same metrics, the following system behaviour is required (steps from above):

1. The receiver always accepts the message except for technical reasons. Especially if a semantic check of the message (which could be done while receiving the message) fails, the receiver will never send a SOAP fault or runtime error (see [Security relevant errors](#)). As long as the message is authentic, can be decrypted and has the correct syntax, it has to be accepted.
2. All business errors due to the accepted message have to be considered in the response. If any business exception occurs, the Receiver (now Sender) has to respond with a [BusinessException](#) data structure with a dedicated error code and error message instead of the regular response
3. With these two messages of points 1. and 2., the communication ends. Most importantly, this means that to any messages of the receiver (regular response or business exception) the originator of the very first message (Sender) will not respond again, although, in theory, problems with internal references to the original message or similar may occur.

This requirement prevents random behaviour of different systems. One system may always process asynchronous messages in a data pipe, others could perform certain semantic checks while receiving the message. These different implementations must not result in different system behaviours in the outside world.

7.4.1.8 Message validation

Common requirements for entire message validation. Consists of

- [Automatic schema validation](#)
- [Communication header and message header consistency](#)
- [Role authorisation](#)
- [Unique identifier for each ION message](#)
- [The reply must have a matching request](#)

7.4.1.9 Automatic schema validation

Automatic schema validation must be enabled. This requirement consists of [Automatic schema validation for incoming messages](#) and [Automatic schema validation for outgoing messages](#).

7.4.1.10 Automatic schema validation for incoming messages

All providers of services have to ensure that the automatic schema validation is enabled while receiving messages. If a schema validation failure occurs, the syntax of the incoming message has the wrong format.

In this case, returning a technical error is allowed, see [Security relevant errors](#).

7.4.1.11 Automatic schema validation for outgoing messages

All consumers of services have to ensure that the automatic schema validation is enabled while sending messages.

If a schema validation failure occurs, the syntax of the outgoing message has the wrong format. In this case, the message must not be sent and a system check has to be done first.

This requirement prevents syntactical wrong messages in the ION communication framework.

7.4.1.12 Communication header and message header consistency

Elements of the communication header (SOAP header) contain a copy of transaction header elements for addressing purposes. This requirement ensures that no inconsistency exists between these message parts. The following elements must have the same value:

- Sender ID
- Sender Role
- Transaction ID
- Transaction timestamp

If this check fails (the data differs in any element), a man-in-the-middle attack could be the reason. Due to this possible security issue, only a short soap fault with the code [E ION DATA OF SOAP HEADER DOES NOT MATCH MESSAGE DATA](#) is sent.

7.4.1.13 The reply must have a matching request

In ION communication, an [ION message with WSS](#) as a request in the business layer must always have a corresponding reply (response or business exception) and vice versa. In order to use the same data structures for synchronous and asynchronous use cases, a reply must always contain the reference to the triggering request.

7.4.1.14 Role authorisation

Certain use cases are only allowed for certain roles.

The processing systems - i.e. the systems at the transport operators and central systems such as the [Hotlist Service](#) - must ensure authorisation. For this purpose, the organisation-role(s) assignments must be regularly requested from a central instance ([Registrar](#), part of the [Scheme Manager](#) system) in order to be able to perform this authorisation.

7.4.1.15 Unique identifier for each ION message

Each ION message must be unique in the whole ION environment.

To achieve this, a system must ensure that the message ID is composed of

- Organisation ID of the system owner
- Role ID of the system owner
- Service (name) which is implemented by the system
- Sequential number

is unique.

7.4.1.16 Use MTOM

The usage of MTOM is obligatory for all business requests and responses which work with WSS. For technical acknowledgements ([deliveryAcknowledgement](#)) concerning the message delivery without WSS, the usage of MTOM is optional.

7.4.1.17 SOAP-Version

The SOAP-version used in the ION for SOAP messages is SOAP 1.1.

The namespace to this version is <http://schemas.xmlsoap.org/soap/envelope>.

7.4.2 Performance

This chapter deals with the aspects of performance. For different scenarios, different performance requirements exist.

This means that communication in a synchronous context has different requirements than communication in an asynchronous context.

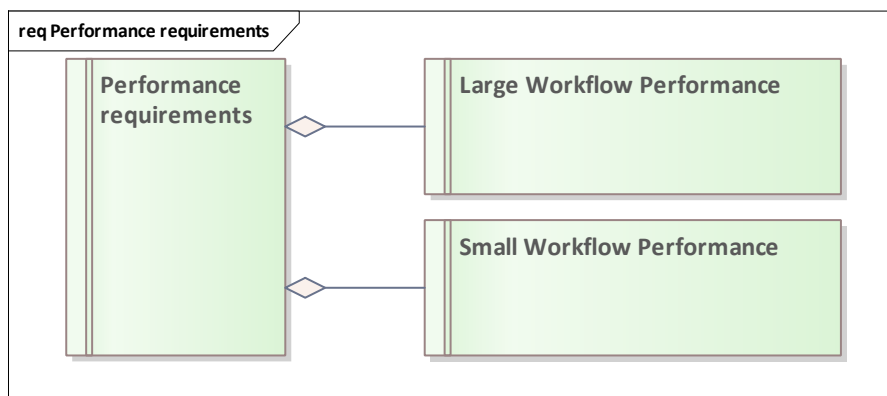


Figure 38: Performance requirements

7.4.2.1 Large Workflow Performance

This requirement defines the maximum response interval for a large workflow (normally if larger lists are used as messages).

To avoid network timeout problems, scenarios with a large workflow must not last longer than the timeout defined in [Appendix: CRE Timeout Configuration Table](#) .

7.4.2.2 Performance requirements

Parent requirement for all requirements which deal with performance.

7.4.2.3 Small Workflow Performance

This requirement defines the maximum response interval for a small workflow (provided by the core systems).

As a user is normally waiting for the result, scenarios with a small workflow must not last longer than defined in [Appendix: CRE Timeout Configuration Table](#).

7.4.3 Scalability

This chapter gives an overview of scalability requirements.

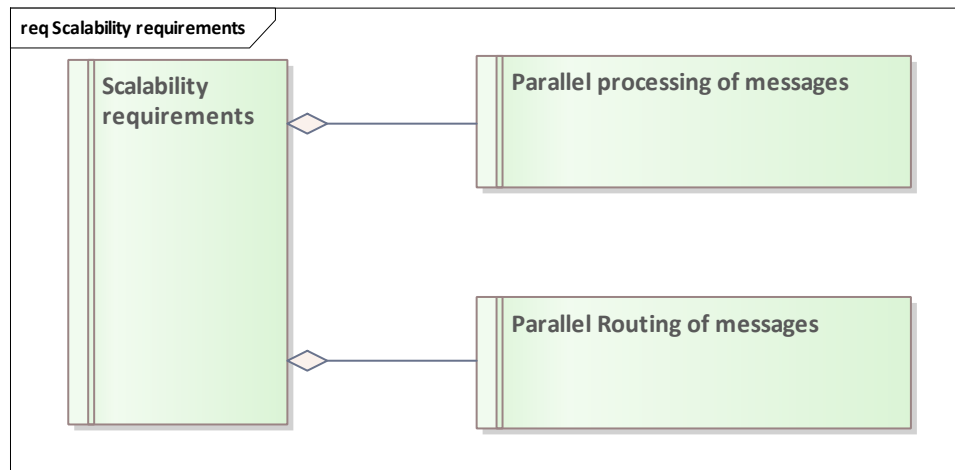


Figure 39: Scalability requirements

7.4.3.1 Parallel processing of messages

Process 5 normal ([Small Workflow Performance](#)) messages per second.

7.4.3.2 Parallel Routing of messages

Route 10 messages with a maximum size of 16 MB at the same time.

7.4.3.3 Scalability requirements

Parent requirement for all requirements which deal with scalability.

7.5 Webservice Security

This chapter describes the WS Policy which is employed in etiCORE. Furthermore, it deals with key management as the foundation for encryption and signing of messages.

The standardised mechanisms of XML-Encryption (XML-Enc, see <https://www.w3.org/TR/xmlenc-core>) and XML-Signature (XML-Sig, see <https://www.w3.org/TR/xmldsig-core>) are used to secure the web services.

The aim is to precisely define the parts to be encrypted and signed when exchanging messages via the ION in the application layer.

XML-Enc is used to achieve [ION:Privacy](#), XML-Sig is used to achieve [ION:Authenticity](#) and thus [ION:Non-Deniability](#) after the message exchange has been completed.

The following chapters describe the individual rules that are defined in the web service policy.

The complete policies, which are also embedded in the respective WSDL, can be found in [Appendix: etiCORE Standard-Policy](#) and [Appendix: etiCORE Empty-Policy](#).

The entire policy specifies that message content between the sending organisation and the receiving organisation is signed and encrypted end-to-end. Only the content of the [SOAP Body](#) is considered.

7.5.1 Key Management

Two levels of cryptographic security are used for the web service infrastructure in the ION.

1. HTTPS TLS transport encryption between the participating system and the CRE.
2. Message-level signatures / encryption of the XML payload data in the SOAP body.

7.5.1.1 Security Levels

Two security levels are relevant for key and certificate management with regard to the ION:

- Level-2: Complete integration test environment with full WSS support (messages may be exchanged without WSS on a test basis, TLS is always mandatory) and own PKI
- Level-3: Production environment with full WSS support (message exchange is only possible with WSS at both transport and message level) and own PKI.

The environments are strictly separated. No keys or certificates may be used for the other environment.

The system instances must be strictly configured and in operation for either level-2 or level-3.

Various checks both in the area of general ION message exchange and in some use cases (there with regard to CV certificates in the area of entitlements and applications) ensure that no level-2 security components can be used in level-3 and vice versa.

For ION messaging, the environments are distinguished by the PKI and the certificates. Each system component that communicates with the ION must be configured to use either level-2 or level-3 as its environment. This is done by embedding the root and sub-CAs in the system's trust store. Verification of the full certificate chain of each ION message ensures that the correct environment is being used.

7.5.1.2 Basic Requirements

For HTTP TLS, RSA key pairs are required for each participant system, where a private key is used on each participant system (per organisation) and a private key on the CRE. Furthermore, each participant system requires the X.509v3 certificate of the CRE which contains its public key. The CRE requires all certificates with the public keys of the participant systems.

It must be ensured that each system can trust the authenticity of the respective certificates. The certificates must be released for use as a TLS client certificate and as a TLS server certificate.

For the message-level signatures/encryption in the SOAP body (see [Message-based encryption \(application layer\)](#)), each communication participant per organisation and role must

- securely store a private key for the signature of the messages to be sent in their system and make a corresponding public key certified by the PKI available in the central directory, accessible via directory service (LDAP for certificate information and OCSP for certificate status)
- securely store a private key for decrypting the messages to be received in their system and make a corresponding PKI-certified public key available in the central directory, accessible via directory service (LDAP for certificate information and OCSP for certificate status)
- have current access to the certificates with the public keys of their communication partners and be able to trust their authenticity

Thus, every organisation with at least one role needs three key pairs (pair = private key and public key):

1. TLS at the organisation level for transport encryption.
2. Encryption/"Enc" per organisation and role for message encryption
3. Signature/"Sig" per organisation and role for message signature

For each additional role, 2 key pairs are added for message encryption and signature.

2048-bit RSA keys are used.

The private keys shall be suitably protected against misuse.

7.5.1.3 Distribution of the Certificates

The participants must generate corresponding RSA key pairs in their systems. The private key remains protected from unauthorised access in the respective system. The public key is registered and certified via CSR in PKCS#10 format with the PKI provided by the Scheme Manager. The PKI responds in PKCS#7 format.

The ION certificates for TLS and WSS can then be requested by the other participants in the PKI via directory service. The PKI offers access via OCSP for status information and LDAP for detailed information.

The combination of the attributes O (organisation ID) and OU (role) can be used here, or alternatively, the key identifier contained by the security header in the SOAP header of the message.

The PKI provides the latest valid certificate (via the "last" pointer) when a request is made using the organisation ID/role.

The certificates are issued by a sub-CA created specifically for the ION for the purpose of message exchange. This sub-CA in turn was issued by a higher-level root CA. The certificates of both CAs are also available in LDAP and can be downloaded to form a certificate chain and loaded into the truststore of the respective system.

The certificate revocation lists are also in LDAP and must be downloaded regularly. The revocation lists have a validity of 24 hours. The parameterisable delay period is 2-4 hours. Thus, the time interval between this update and the next update is 26-28 h according to [RFC 5280 \[6\]](#).

Furthermore, the PKI supports OCSP for checking the certificates status.

A certificate may only be used if at least one of the procedures for checking certificate revocations (revocation list or OCSP) provides current data and the certificate is not marked as revoked according to this data at the time of use (signature check or encryption).

The authenticity of the keys is checked by the individual systems themselves via the certification hierarchy up to the root certificate.

7.5.1.4 Structure of the Certificates

The profiles of the three certificate types (for TLS authentication and message-based signature and encryption) are structured as follows.

- Version: Version 3 (X.509) is used
- Subject:
 1. field CN: identifier in LDAP
 - for encryption and signature:** assigned by PKI when submitting a CSR: [organisation ID in hex]-[role]-[use:enc,sig]-[sequence number below organisation,(role),purpose]. E.g. 15E0-05-sig-0007
 - for TLS:** URL where the TLS key is placed, normally the domain of the web-server in the DMZ. E.g. www.mywebserver-domain.com
 2. field O: Organisation ID of the organisation
 3. field OU: role of the organisation (not for TLS certificate)
 4. field C: country (two-letter-code, ISO 639)
- Issuer: identification of the body (sub-CA) that issued the certificate.
- Serial Number: unique ID of the certificate
- Valid from: date from which the certificate is valid.
- Valid Until: date when the certificate expires
- Public Key Algorithm ID: ID of the algorithm used in the certified key: RSA 2048 bit
- Public Key: the public key itself
- Key Usage:
 1. message-based encryption method,
 2. message-based signature procedure or
 3. TLS: authentication method with session key agreement
- - Certificate Signature Algorithm: (ID of the signature algorithm used by the CA to generate the certificate): RSASSA-PSS
- - Certificate Signature: the certificate signature

Note: the maximum validity period of the certificates is 3 years.

7.5.1.5 Validity of Certificates

For each certificate used

- the validity period of the certificate must contain the time of the signature verification
- there must either be a valid OCSF response for the certificate at the time of signature verification in which the certificate is not revoked
- alternatively, there must be a valid revocation list for the certificate at the time of the signature verification in which the certificate is not revoked

The OCSF request is made online directly to the PKI system. Alternatively, the certificate revocation list can be downloaded regularly (at least once a day) and checked for an entry of the certificate currently in use before establishing a connection or exchanging messages. After the ION certificate expires, a new key pair is generated, and the new public key is certified. This means that keys for expired certificates are no longer used and are unknown to the system.

7.5.1.6 Transition Phase for Certificate Renewal

The certificates must be renewed regularly according to the [Validity of Certificates](#). To ensure an uninterrupted, continuous exchange of messages based on [ION:Security](#), certificates must be renewed before their validity expires. Certificates must be renewed at least 6 weeks before they expire. As the systems usually cache the certificates, the old certificate must not be marked as invalid. Instead, the systems must be able to work with both variants during the transition period. A new key pair is generated for the new certificate validity period and the new public key is certified. This means that both the old and the new public key may be referenced in the security header of the SOAP message in the WSS during the transition period. Expired certificates must be automatically removed from the cache of the systems. This means that if the old certificate has expired, messages may only be accepted for the new certificate, which is then already in use.

7.5.2 Message-based encryption (application layer)

In addition to transport security, all user data in the ION is always transmitted encrypted and signed by the sender.

This protection of the user data takes place end-to-end, i.e. the sender system encrypts the user data of the messages; the user data thus remain inaccessible to systems that merely mediate (especially the CRE).

Data decryption and signature checking are only performed in the target system.

Data that serves to mediate the user data must accordingly be made explicitly available to the routing engines, especially the CRE.

Web service security ([\[8\] WS-Security 1.1](#)) is used as the procedure for end-to-end encryption. The user data of the exchanged messages is placed in the body of the web service messages. The routing information is found in the header of the web service messages. Accordingly, the SOAP body is signed and encrypted, while the SOAP header is transmitted unencrypted.

Due to the condition that transport encryption is prescribed in the ION and that the CRE requires the transport encryption certificate to match the corresponding routing information, it is not possible to attack the routing information in the SOAP header from outside the ION. The organisations participating in the ION as well as the ION access points are trusted with regard to routing. Therefore, the SOAP header is not signed at the message level. Important details of the WS-Security parameters used are defined below. A Web Service Policy corresponding to the specified parameters is attached in [Appendix: etiCORE Standard-Policy](#).

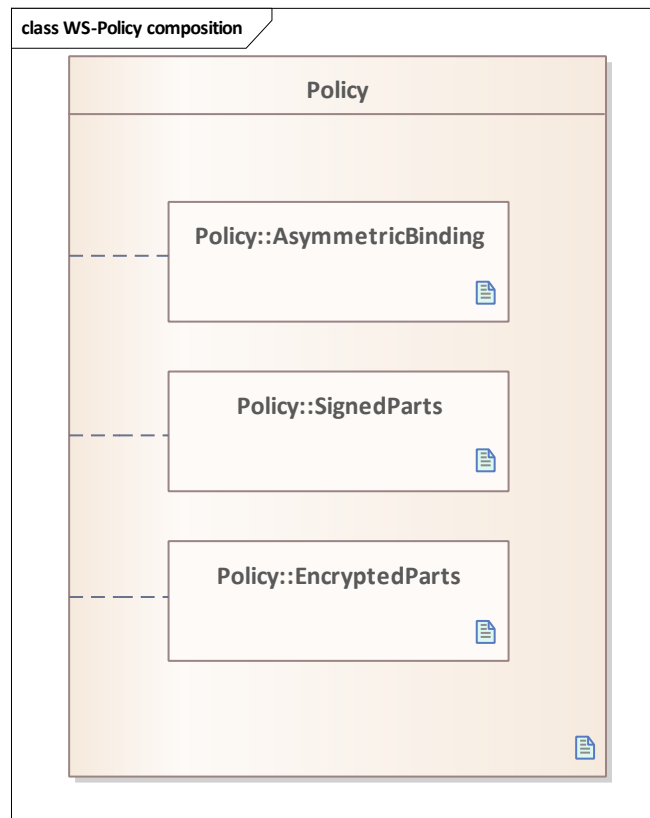


Figure 40: WS-Policy composition

A diagram that shows the most important parts of a WSS [Policy](#). The employment of these parts will be explained in the following chapters.

7.5.2.1 Algorithm Suite

To achieve maximum compatibility the algorithm suite **Basic256Sha256** is employed, see also [Appendix: etiCORE Standard-Policy](#) and [\[9\] WS-SecurityPolicy v1.2](#).

```
<sp:AlgorithmSuite>
  <wsp:Policy>
    <sp:Basic256Sha256/>
  </wsp:Policy>
</sp:AlgorithmSuite>
```

7.5.2.2 Sender Certificate

For the sender, X.509v3 certificates are used.

It is assumed that the certificates were exchanged out-of-band.

Therefore, the certificates are not embedded in the SOAP messages, but only referenced. The reference to the certificate is given in the form **Issuer-Serial**.

See also [Appendix: etiCORE Standard-Policy](#) and [\[9\] WS-SecurityPolicy v1.2](#).

```
<!-- Client X.509-Certificate -->
<sp:InitiatorToken>
  <wsp:Policy>
    <!-- do NOT include the X509 certificate data in the request,
```

```
        provide issuer reference only -->
<sp:X509Token sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-
securitypolicy/200702/IncludeToken/Never">
  <wsp:Policy>
    <sp:WssX509V3Token10/>
    <sp:RequireIssuerSerialReference/>
  </wsp:Policy>
</sp:X509Token>
</wsp:Policy>
</sp:InitiatorToken>
```

7.5.2.3 Receiver Certificate

For the receiver, X.509v3 certificates are used.

It is assumed that the certificates were exchanged out-of-band.

Therefore, the certificates are not embedded in the SOAP messages, but only referenced. The reference to the certificate is given in the form **Issuer-Serial**.

See also [Appendix: etiCORE Standard-Policy](#) and [\[9\] WS-SecurityPolicy v1.2](#).

```
<!-- Server X.509-Certificate -->
<sp:RecipientToken>
  <wsp:Policy>
    <!-- do NOT include the X509 certificate data in the request,
         provide issuer reference only -->
    <sp:X509Token sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-
securitypolicy/200702/IncludeToken/Never">
      <wsp:Policy>
        <sp:WssX509V3Token10/>
        <sp:RequireIssuerSerialReference/>
      </wsp:Policy>
    </sp:X509Token>
  </wsp:Policy>
</sp:RecipientToken>
```

7.5.2.4 SOAP Header Layout

For the SOAP-Header, the convention **Strict** is used.

See [\[9\] WS-SecurityPolicy v1.2](#), Section 6.7.

```
<sp:Layout>
  <wsp:Policy>
    <!-- Use strict Header Layout -->
    <sp:Strict/>
  </wsp:Policy>
</sp:Layout>
```

7.5.2.5 WSS Timestamp

The message timestamp **IncludeTimestamp** according to WS-Security is used. However, the message validity period must be extended for asynchronous messages.

A default validity period of, for example, 5 minutes would result in messages delivered via store-and-forward due to the unavailability of the target system being discarded as too old. A compromise must be found here between the possibility of replay attacks and the desire for a store-and-forward procedure.

This specification sets a message validity of 7 days. For this, the implementations of the individual web service security stack manufacturers must be configured. The policy attached under [Appendix: etiCORE Standard-Policy](#) only ensures that the timestamp is transmitted in the message. There are two parties involved in the validity period of the message:

1. The evaluation of this timestamp under consideration of the maximum validity period is the responsibility of the receiving systems (under restriction, see below).

2. The expiry date of the message is set directly by the client when sending the message.

Unfortunately, the XSD (see <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd>) is ambiguous as a binding structure in Timestamp because both the timestamp for generating the message and the timestamp for the expiry date are optional. According to this specification, both values must be set in practice, so that the content according to the following example is to be assumed:

```
<wsu:Timestamp xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" wsu:Id="XWSSGID-12901552723161304893044">
  <wsu:Created>2017-08-01T08:30:50Z</wsu:Created>
  <wsu:Expires>2017-08-08T08:30:50Z</wsu:Expires>
</wsu:Timestamp>
```

The example shows that there are 7 days between the creation of the message and the expiry. So, in this example, the sending system would set the values and the receiving system would evaluate the value in *Expires*.

If the client's implementation is incorrect (e.g. the default value of 5 minutes is used), the receiving system will send a SOAP Fault that the message has expired.

In most implementations of the web service security stacks, the message is automatically rejected, so that a different behaviour can only be implemented with a great deal of programmatic effort (if at all).

Therefore, for asynchronous messages, the sending system must be responsible for setting the expiry date correctly.

If *Expires* is not set (which is unfortunately possible due to the XSD), it is recommended that the receiving system additionally evaluates *Created* with an offset of 7 days to the timestamp of the creation of the message.

It should be noted that an extension of the expiry time only makes sense for asynchronous messages. The systems must therefore distinguish between synchronous and asynchronous messages with regard to the expiry time. For synchronous messages without possible intermediate storage, the value can remain at the default of 5 minutes.

```
<!-- Provide creation date and expiry in request -->
<!-- Note: Standard expiry of 5 minutes must be increased to 7 days
      for asynchronous requests -->
<sp:IncludeTimestamp/>

<!-- extracted example, see also Security header -->
<wsu:Timestamp xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" wsu:Id="XWSSGID-12901552723161304893044">
  <wsu:Created>2017-08-01T08:30:50Z</wsu:Created>
  <wsu:Expires>2017-08-08T08:30:50Z</wsu:Expires>
</wsu:Timestamp>
```

7.5.2.6 Signature Scope

The signature scope is the [SOAP Body](#) and its content. The [SOAP Body](#) always contains the payload in a common form of [request payload](#), [response payload](#) or [exception payload](#).

The tag *IncludeTimestamp* adds the timestamp to the header which leads to the timestamp being signed as well even though it is part of the header and even though the signed parts are specified to be the body only.

```
<sp:IncludeTimestamp />
...

<!-- sign message body only -->
<sp:SignedParts>
  <sp:Body />
</sp:SignedParts>
```

7.5.2.7 Encryption Scope

The encryption scope is the [SOAP Body](#) and its content. The [SOAP Body](#) always contains the payload in a common form of [request payload](#), [response payload](#) or [exception payload](#). The tag *EncryptSignature* leads to the signature being encrypted as well even though it is part of the header and even though the encrypted parts are specified to be the body only.

```
<sp:EncryptSignature/>

...

<!-- encrypt message body only -->
<sp:EncryptedParts>
  <sp:Body />
</sp:EncryptedParts>
```

7.5.2.8 Embedded binary Data

Binary data is embedded in the XML format by means of the MTOM (see <https://www.w3.org/TR/soap12-mtom>) procedure and, thus, covered by signature and encryption.

```
<!-- Messages from the web service MUST be optimised using MTOM -->
<wsoma:OptimizedMimeSerialization />
```

7.5.3 Appendix: etiCORE Standard-Policy

The standard policy that describes the encryption and signature of all synchronous operations (request and response) as well as requests of asynchronous operations.

Additionally, the [OptimizedMimeSerialization-tag](#) determines that the message transfer is done using MTOM.

This policy is called *etiCOREStandardPolicy* and is integrated into all WSDLs.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsp:Policy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/ns/ws-policy http://www.w3.org/2007/02/ws-policy.xsd"
  xmlns:wsp="http://www.w3.org/ns/ws-policy" xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-
securitypolicy/200702"
  xmlns:wsoma="http://schemas.xmlsoap.org/ws/2004/09/policy/optimizedmimeserialization"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
  targetNamespace="http://www.w3.org/ns/ws-policy" wsu:Id="etiCOREStandardPolicy">
  <wsp:ExactlyOne>
    <wsp>All>
      <!-- Responses from the Web service MUST be optimized using MTOM -->
      <wsoma:OptimizedMimeSerialization/>
      <sp:AsymmetricBinding>
        <wsp:Policy>
          <!-- Client X.509-Certificate -->
          <sp:InitiatorToken>
            <wsp:Policy>
              <!-- do NOT include the X509 certificate data in the request,
                provide issuer reference only -->
              <sp:X509Token sp:IncludeToken=
                "http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/Never">
                <wsp:Policy>
                  <sp:WssX509V3Token10/>
                  <sp:RequireIssuerSerialReference/>
                </wsp:Policy>
              </sp:X509Token>
            </wsp:Policy>
          </sp:InitiatorToken>
        </wsp:Policy>
      </sp:AsymmetricBinding>
    </wsp>All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

```
        </sp:X509Token>
      </wsp:Policy>
    </sp:InitiatorToken>
    <!-- Server X.509-Certificate -->
    <sp:RecipientToken>
      <wsp:Policy>
        <!-- do NOT include the X509 certificate data in the request,
              provide issuer reference only -->
        <sp:X509Token sp:IncludeToken=
          "http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/Never">
          <wsp:Policy>
            <sp:WssX509V3Token10/>
            <sp:RequireIssuerSerialReference/>
          </wsp:Policy>
        </sp:X509Token>
      </wsp:Policy>
    </sp:RecipientToken>
    <sp:Layout>
      <wsp:Policy>
        <!-- Use strict Header Layout -->
        <sp:Strict/>
      </wsp:Policy>
    </sp:Layout>
    <!-- Provide creation date and expiry in the request -->
    <!-- Note: Standard expiry of 5 minutes must be increased to 7 days
          for asynchronous requests -->
    <sp:IncludeTimestamp/>
    <sp:OnlySignEntireHeadersAndBody/>
    <sp:AlgorithmSuite>
      <wsp:Policy>
        <sp:Basic256Sha256/>
      </wsp:Policy>
    </sp:AlgorithmSuite>
    <sp:EncryptSignature/>
  </wsp:Policy>
</sp:AsymmetricBinding>
<!-- WSS 1.0 options-->
<sp:Wss10>
  <wsp:Policy>
    <sp:MustSupportRefIssuerSerial/>
  </wsp:Policy>
</sp:Wss10>
<!-- sign and encrypt message body only -->
<sp:SignedParts>
  <sp:Body/>
</sp:SignedParts>
<sp:EncryptedParts>
  <sp:Body/>
</sp:EncryptedParts>
</wsp>All>
</wsp:ExactlyOne>
</wsp:Policy>
```

7.5.4 Appendix: etiCORE Empty-Policy

This policy optionally allows only MTOM via OptimisedMimeSerialization tag, otherwise the messages are neither signed nor encrypted. This policy is only used for the replies/responses of asynchronous operations ([deliveryAcknowledgement](#)) and is accordingly included in all WSDLs that offer at least one asynchronous operation.

With the help of this policy, it can be specified at the message level that responses are sent in plain text.

This policy is called *etiCOREEmptyPolicy*.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsp:Policy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/ns/ws-policy http://www.w3.org/2007/02/ws-policy.xsd"
  xmlns:wsp="http://www.w3.org/ns/ws-policy"
  xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
  xmlns:wsoma="http://schemas.xmlsoap.org/ws/2004/09/policy/optimizedmimeserialization"
```



```
targetNamespace="http://www.w3.org/ns/ws-policy" wsu:Id="etiCOREEmptyPolicy">
```

```
<!-- This policy is employed for the non-encrypted deliveryAcknowledgement response messages in asynchronous services and does not cause a security header in the message. This is important since the ESB sends a plain response without any security header (in case of message buffering) and this must be compatible with the response which is sent directly by the end-point system. -->
```

```
<wsp:ExactlyOne>
```

```
  <wsp:All>
```

```
    <!-- Optionally allow MTOM in messages -->
```

```
    <wsoma:OptimizedMimeSerialization wsp:Optional="true" />
```

```
  </wsp:All>
```

```
</wsp:ExactlyOne>
```

```
</wsp:Policy>
```

8 etiCORE-Model API

8.1 ion

8.1.1 BusinessAcknowledgement

Data structure that contains an acknowledgement of a previous ION request message, together with communication information for the current message. Additionally and optionally, a list of (processing) warnings can be attached.

8.1.2 BusinessException

Data structure that transports a business exception. Consists of a reference to a previous request, which processing caused this exception, an optional warning list with further information, if needed and the error itself, which comes with an abbreviation and an optional error message.

Attributes
error:Error
Error which raised due to a previous request.
Multiplicity:

8.1.3 CommunicationInformation

Base type contained in all ION messages. Contains sender, receiver and message information.

Attributes
messageId:IonMessageId
Message ID. See the documentation of the referenced type
Multiplicity:
messageTimestamp:ZonedDateTime
Timestamp when the message was sent via the ION by the originator (also adapted in retry scenarios).
Multiplicity:
processInstanceId:ProcessInstanceId
ID of the process instance containing the name of the basic process and a UUID.

Multiplicity:
receiverId:OrganisationId Organisation ID of the message receiver Multiplicity:
receiverRole:PartnerRoleCode Electronic fare management (EFM) role of the message receiver Multiplicity:
receiverService:ServiceName Service of the message receiver Multiplicity:
repeatCounter:RepeatCounter Number of times this message has already been sent. Defaults to zero. Multiplicity: [0..1]

8.1.4 CompactCommunicationInformation

Data structure which contains a reduced communication information for the current message: Only the message number and the processInstanceId are present.

Attributes
messageNumber:IonMessageNumber Multiplicity:
processInstanceId:ProcessInstanceId Multiplicity:

8.1.5 CompactMessage

Base type for all compact business-level request, response, and exception types.

Attributes

compactCommunicationInformation:CompactCommunicationInformation

Compact communication information.

Multiplicity:

8.1.6 CompactRequest

Base type for all types that are used in compact requests (on the business level).

Asynchronously communicated replies are requests (on the technical level), but replies (on the business level).

8.1.7 CompressedBusinessReplyList

The structure that contains a compressed list with business acknowledgements or business exceptions. Both kinds of list entries are possible.

Attributes

compressedBusinessReplyList:CompressedXmlList

Contains the compressed list

Multiplicity:

8.1.8 CompressedXmlList

This element contains a Base64-encoded zipped XML list. To get the (unzipped) XML format for later processing purposes, you have to decode the Base64 string to binary and then unpack the resulting zip file with a standard zip program or programming API.

8.1.9 DiscardedMessages

List of discarded message metadata information.

Attributes

discardedMessagesMetadata:DiscardedMessagesMetadata

List of discarded message metadata information. Contains at least one element.

Multiplicity: [1..unbounded]

8.1.10 DiscardedMessagesMetadata

Metadata of messages which become discarded due to WSS expiry. Used to inform the original sender that the message could not be delivered to the recipient. Since the number of messages can be very high, this structure gives an overview but no information about single messages. Furthermore, due to security reasons, the original message would be refused by the WSS framework.

Attributes

amountOfExpiredMessages:integer

This field contains the amount of messages for the same receiver, role, service, and operation which were discarded since they couldn't be delivered within the configured time period.

Multiplicity:

amountOfUnexpectedReplies:integer

This field contains the amount of messages for the same receiver, role, service, and operation which were discarded because the target system replied in an unexpected way.

Multiplicity:

fromMessageTimestamp:ZonedDateTime

The timestamp of the oldest message used for metadata information grouping.

Multiplicity:

operationName:OperationName

The name of the operation as message type as information which messages could not be delivered.

Multiplicity:

receiverId:OrganisationId

The organisation ID of the receiver that was intended to get the message.

Multiplicity:

receiverRole:PartnerRoleCode

The (EFM) partner role of the receiver that was intended to get the message.

Multiplicity:

receiverService:ServiceName

The name of the service that the original message was intended for.

Multiplicity:

untilMessageTimestamp:ZonedDateTime

The timestamp of the newest message used for metadata information grouping.

Multiplicity:

8.1.11 Error

Classifies an event as an error level.

8.1.12 Event

Common abstract type for all possible events.

Attributes
<p>applicationReference:AppInstanceId</p> <p>Optional reference to one or more application instance IDs this event relates to. This application can be a user medium application or motics app. As the SAM ID has the type of appInstanceId, applicationReference may indicate a SAM, in case of SAM related cases such as ConfigureSamException.</p> <p>Multiplicity: [0..unbounded]</p>
<p>entitlementReference:EntitlementId</p> <p>Optional reference to one or more Entitlement IDs this event relates to.</p> <p>Multiplicity: [0..unbounded]</p>
<p>eventDescription:EventDescription</p> <p>Description that explains the event and gives further details.</p> <p>Multiplicity: [0..1]</p>
<p>eventIdentifier:EventIdentifier</p> <p>String-based identifier which specifies the event. The identifier gives a first impression of the event. Must be one of the identifiers which are defined either globally or service-wide as (\$area[Warning Error]Enum).</p> <p>Multiplicity:</p>
<p>messageReference:IonMessageId</p> <p>Optional reference to one or more ION message IDs that specifies the previous message(s) related to this event.</p> <p>Multiplicity: [0..unbounded]</p>

8.1.13 EventDescription

Additional description for error code inside a common rejection message.

8.1.14 IonMessageId

The data type of an ID for messages which are processed via the ION. The conversation has a wider scope than the message since the sender references its process with this ID as well as the receiver, which has to process the message data. This element serves as a correlation between the sender's request and the receiver's reply. The message ID is an ION-wide unique combination of the message number, the sender service, the sender's organisation ID and the sender's role.

Attributes
messageNumber:IonMessageNumber
Multiplicity:
senderId:OrganisationId
Multiplicity:
senderRole:PartnerRoleCode
Multiplicity:
senderService:ServiceName
Multiplicity:

8.1.15 IonMessageNumber

The message number of an ION message. Must be unique for one sender (organisation ID), role and service. A sender is not allowed to use a message number again (exception: a deliberate retry in the case of a communication error).

8.1.16 Message

Base type for all business-level request, response and exception types. Note that delivery acknowledgements / rejections are not part of the hierarchy.

Attributes
communicationInformation:CommunicationInformation
Communication information.
Multiplicity:

8.1.17 OperationName

Defines the name of the target operation only for routing purposes. The name is identical to the (encrypted) payload name. The web service framework itself ensures that the right operation is invoked for the operation name, so it is not needed for operation invocation identification as soon as the payload has been decrypted. The operation name is needed to determine the context (synchronous, asynchronous, timeout, cachable, etc.).

8.1.18 OriginalRequestCommunicationInformation

Base data for all reply messages. Contains the message ID of the request for correlation purposes.

Attributes
originalRepeatCounter:RepeatCounter Contains the repeat counter of the original request for operational monitoring purposes. Multiplicity: [0..1]
originalRequestCorrelationId:IonMessageId Contains the message ID of the original request for correlation purposes. Multiplicity:
originalRequestTimestamp:ZonedDateTime Contains the time stamp of the original request for plausibility check purposes. Multiplicity:

8.1.19 ProcessInstanceId

ID of the process instance containing the name of the basic process and a UUID. To generate a process instance ID, take the basic process name (from ProcessNameEnum) and a UUID (in canonical string form) and combine them separated by an underscore. Template:
ProcessName_00000000-0000-0000-0000-000000000000

8.1.20 RepeatCounter

Number of times this message has already been sent.

8.1.21 Reply

Base type for all types that are used in replies (on the business level). Asynchronously communicated replies are requests (on the technical level), but replies (on the business level).

Attributes
originalRequestCommunicationInformation:OriginalRequestCommunicationInformation

Original request communication information.

Multiplicity:

warningList:WarningList

List of events that might happen during processing. Events can be additional information or warnings, which might refer to previous messages (not only one previous request).

Multiplicity: [0..1]

8.1.22 Request

Base type for all types that are used in requests (on the business level). Asynchronously communicated replies are requests (on the technical level), but replies (on the business level).

8.1.23 Warning

Classifies an event as a warning level.

8.1.24 WarningList

List of warnings.

Attributes

warning:Warning

Multiplicity: [1..unbounded]

8.2 ion-enums

8.2.1 ServiceName

Defines the abstract name of the service for routing purposes. See ServiceNameEnum for allowed values.

8.2.2 ProcessNameEnum

Matches the basic process in most cases, but may deviate at some points. E.g. the inspection process may produce several EntitlementInspectedNotifications, which would all get their own Process Instance ID sharing the ProcessName "InspectEntitlement".

Enumeration values

Value: AddAcceptanceEntryToHotlistConfiguration

Description:
Value: AutoloadStoredValuePaymentMethod Description:
Value: BlockApplication Description:
Value: BlockEntitlement Description:
Value: CancelOrder Description:
Value: ChangeUserTariffParameters Description:
Value: ChargeAccountBasedPaymentMethod Description:
Value: ConfigureUserMedium Description:
Value: CreditAccountBasedPaymentMethod Description:
Value: CreditStoredValuePaymentMethod Description:
Value: DebitAccountBasedPaymentMethod Description:
Value: DebitStoredValuePaymentMethod Description:
Value: DiscardMessages Description:
Value: DistributeActionListRetrievalConfiguration Description:
Value: ExecuteOrderedEntitlementBlocking Description:

Value: ExecuteOrderedEntitlementIssuance Description:
Value: ExecuteOrderedEntitlementTermination Description:
Value: ExecuteOrderedEntitlementUnblocking Description:
Value: ExecuteSamConfigurationScript Description:
Value: ExecuteSamResetScript Description:
Value: GetEntitlementsOfUserMedium Description:
Value: HandleDefectiveUserMedium Description: Process scope is from terminal to own back-office system. A potential lookup of the app instance ID and the hotlisting process get their own process instance IDs.
Value: HandleObsoleteOrder Description:
Value: HotlistApplication Description:
Value: HotlistAuthenticationKey Description:
Value: HotlistEntitlement Description:
Value: HotlistOrganisation Description:
Value: HotlistSam Description:
Value: InspectEntitlement Description: The basic process is called "Inspect user medium with application". This basic process may produce several EntitlementInspectedNotifications, which would all get their own Process Instance ID sharing the ProcessName "InspectEntitlement".

Value: InspectStaticEntitlement Description: The basic process is called "Inspect user medium without application". This basic process may produce several StaticEntitlementInspectedNotifications, which would all get their own Process Instance ID sharing the ProcessName "InspectStaticEntitlement".
Value: IssueEntitlement Description:
Value: IssueStaticEntitlement Description:
Value: LogInvalidApplication Description:
Value: LogInvalidEntitlement Description:
Value: LookupApplicationInstanceId Description:
Value: LookupSAMOwner Description:
Value: NotifyEvents Description:
Value: OrderEntitlementBlocking Description:
Value: OrderEntitlementIssuance Description:
Value: OrderEntitlementTermination Description:
Value: OrderEntitlementUnblocking Description:
Value: OrderGroup Description:
Value: PersonaliseApplication Description:
Value: ProcessCheckinNotificationList

Description:
Value: ProcessCheckoutNotificationList Description:
Value: ProcessEntitlementInspectedNotificationList Description:
Value: ProcessEntitlementIssuedNotificationList Description:
Value: ProcessNewInformationAboutCustomerAndDiscounts Description:
Value: ProcessStaticEntitlementInspectedNotificationList Description:
Value: ProcessStaticEntitlementIssuedNotificationList Description:
Value: RechargeStoredValuePaymentMethod Description:
Value: RecordEntitlementWithinCheckInProcess Description:
Value: RecordEntitlementWithinCheckOutProcess Description:
Value: ReimburseStoredValuePaymentMethod Description:
Value: RemoveAcceptanceEntryFromHotlistConfiguration Description:
Value: RemoveAuthenticationKeyFromHotlist Description:
Value: RemoveOrganisationFromHotlist Description:
Value: RemoveProductAcceptanceFromParticipants Description:

Value: RetrieveAndDistributeActionList Description:
Value: RetrieveApplicationHotlist Description:
Value: RetrieveAuthenticationKeyHotlist Description:
Value: RetrieveCaCertificateRepository Description:
Value: RetrieveCvCertificateRevocationList Description:
Value: RetrieveEntitlementHotlist Description:
Value: RetrieveOrganisationHotlist Description:
Value: RetrieveOrganisationList Description:
Value: RetrieveProductAcceptanceConfigurationList Description:
Value: RetrieveSamHotlist Description:
Value: RetrieveUnclaimedListInformation Description:
Value: RevokeApplicationHotlisting Description:
Value: RevokeEntitlementHotlisting Description:
Value: RevokeSamHotlisting Description:
Value: SetServiceAvailable Description:

Value: SetServiceUnavailable Description:
Value: TerminateApplication Description: The basic process is called "Take back application". This basic process may produce additional notifications regarding entitlement termination, crediting or reimbursement, which would all get their own Process Instance ID.
Value: TerminateEntitlement Description: The basic process is called "Take back entitlement". This basic process may produce an additional notification regarding crediting or reimbursement, which would get its own Process Instance ID.
Value: TerminateStaticEntitlement Description: The basic process is called "Tack back static entitlement".
Value: UnblockApplication Description:
Value: UnblockEntitlement Description:
Value: UpdateTerminalHotlists Description:
Value: UpdateTerminalTariffModules Description:
Value: ValidateEntitlement Description:

8.2.3 ServiceNameEnum

Defines the abstract name of the service for routing purposes.

Enumeration values
Value: ccp Description: Customer Contract Partner Standard Service
Value: ccp-oa-execution Description: Customer Contract Partner Action Execution Service
Value: ccp-oa-ordering Description: Customer Contract Partner Action Ordering Service

Value: ccp-ste Description: Customer Contract Partner Static Entitlement Service
Value: cre Description: Central Routing Engine Standard Service
Value: esh Description: (((eTicket Security Hub service (as part of the Scheme Manager)
Value: hotlist Description: Hot List Standard Service
Value: mms Description: Media management system (as part of the Scheme Manager)
Value: po Description: Product Owner Standard Service
Value: po-oa-management Description: Product Owner Action List Service
Value: po-ste Description: Product Owner Static Entitlement Service
Value: so Description: Service Operator Standard Service
Value: so-ste Description: Service Operator Static Entitlement Service

8.3 ion-event-enums

8.3.1 IonErrorEnum

ION error codes for ION-related errors.

Enumeration values
Value: E_ION_DATA_OF_SOAP_HEADER_DOES_NOT_MATCH_MESSAGE_DATA Description: The SOAP header does not match the message data.
Value: E_ION_DUPLICATE_ION_MESSAGE_ID Description: A message with the same ION message ID has already been received.

Value: E_ION_DUPLICATE_RESPONSE Description: The original request has already a correlation to a previous response.
Value: E_ION_OPERATION_NOT_IMPLEMENTED Description: The operation specified by the sender is not implemented by the receiver.
Value: E_ION_ORIGINAL_REQUEST_NOT_FOUND Description: The response contains a reference to an original request which does not exist.
Value: E_ION_RECEIVER_ORGANISATION_MISMATCH Description: The transmitted Organisation ID of the request does not match the Organisation ID of the receiver.
Value: E_ION_RECEIVER_ROLE_MISMATCH Description: The transmitted Role ID of the request does not match the Role ID of the receiver.
Value: E_ION_RECEIVER_SERVICE_MISMATCH Description: The transmitted service name of the request does not match the service name of the receiver.
Value: E_ION_SAME_MESSAGE_IN_PROGRESS Description: The system has found an identical message that is being processed. This can happen if several identical messages have been stored in the CRE, which are now sent together to the target system with only a few milliseconds between them.
Value: E_ION_UNKNOWN_SENDER Description: Check whether the Organisation-ID is listed in the list of Organisations that can be obtained from the Scheme Manager.
Value: E_ION_UNKNOWN_TECHNICAL_PROBLEM Description: Unknown technical Problem.
Value: E_ION_WRONG_SENDER_ROLE Description: Wrong sender role.
Value: E_ION_WRONG_SENDER_SERVICE Description: Wrong sender service.

8.3.2 IonWarningEnum

8.4 ion-event-types

8.4.1 E_ION_DATA_OF_SOAP_HEADER_DOES_NOT_MATCH_MESSAGE_DATA

The SOAP header does not match the message data.

8.4.2 E_ION_DUPLICATE_ION_MESSAGE_ID

A message with the same ION message ID has already been received.

8.4.3 E_ION_DUPLICATE_RESPONSE

The original request has already a correlation to a previous response.

8.4.4 E_ION_OPERATION_NOT_IMPLEMENTED

The operation specified by the sender is not implemented by the receiver.

8.4.5 E_ION_ORIGINAL_REQUEST_NOT_FOUND

The response contains a reference to an original request which does not exist.

8.4.6 E_ION_RECEIVER_ORGANISATION_MISMATCH

The transmitted Organisation ID of the request does not match the Organisation ID of the receiver.

8.4.7 E_ION_RECEIVER_ROLE_MISMATCH

The transmitted Role ID of the request does not match the Role ID of the receiver.

8.4.8 E_ION_RECEIVER_SERVICE_MISMATCH

The transmitted service name of the request does not match the service name of the receiver.

8.4.9 E_ION_SAME_MESSAGE_IN_PROGRESS

The system has found an identical message that is being processed. This can happen if several identical messages have been stored in the CRE, which are now sent together to the target system with only a few milliseconds between them.

8.4.10 E_ION_UNKNOWN_SENDER

Check whether the Organisation-ID is listed in the list of Organisations that can be obtained from the Scheme Manager.

8.4.11 E_ION_UNKNOWN_TECHNICAL_PROBLEM

Unknown technical Problem.

8.4.12 E_ION_WRONG_SENDER_ROLE

Wrong sender role.

8.4.13 E_ION_WRONG_SENDER_SERVICE

Wrong sender service.

8.5 ion-communication

8.5.1 DeliveryAcknowledgement

Defines the structure of a common acknowledgement message. Empty. Always used in the asynchronous context. This message can be sent from the central routing engine (CRE) or from an endpoint target system and indicates that a message has been delivered.

Attributes
deliveredTo:DeliveredToEnum
Multiplicity:

8.5.2 deliveryAcknowledgement

Technical acknowledgement message for asynchronous communication. Used in WSDLs.

8.5.3 DeliveryRejection

Defines the structure of a common acknowledgement message in case of a delivery rejection. Always used in the asynchronous context. This message can be sent from the central routing engine (CRE) or from an endpoint target system. The client may distinguish it by evaluating if the error code was set by the CRE or the endpoint system.

Attributes
deliveredTo:DeliveredToEnum
Multiplicity:
errorDescription:EventDescription

Multiplicity: [0..1]

errorIdentifier:IonCommunicationErrorCode

Multiplicity:

8.5.4 deliveryRejection

Technical rejection message for asynchronous communication. Used in WSDLs.

8.5.5 InterfaceVersion

The supported version of the interface as implemented in the sender system, e.g., "3.7.2". The relevant information for the supported etiCORE version is the supported etiCORE version of the participant's back-office system, not the supported etiCORE version of the JSB, ION adapter, etc.

8.5.6 IonRoutingHeader

The structure that defines the content of an unencrypted soap header for routing purposes. Offers full routing information for SOAP requests

Attributes

interfaceVersion:InterfaceVersion

Multiplicity:

messageNumber:IonMessageNumber

Part of the message ID in the message payload.

Multiplicity:

operationName:OperationName

The name of the operation to be called. Used for routing.

Multiplicity:

optionalRoutingInfo:OptionalRoutingInfo

Optional field for additional routing information which is not evaluated by the central routing engine, but by local routing engines employed in a transport association consisting of several public transport companies.

Multiplicity: [0..1]
<p>processInstanceId:ProcessInstanceId</p> <p>ID of the process instance containing the name of the basic process and a UUID.</p> <p>Multiplicity:</p>
<p>receiverId:LengthRestrictedOrganisationId</p> <p>The organisation ID of the message's receiver. Used for routing.</p> <p>Multiplicity:</p>
<p>receiverRole:PartnerRoleCode</p> <p>The (EFM) partner role of the receiver. Used for routing.</p> <p>Multiplicity:</p>
<p>receiverService:ServiceName</p> <p>The (abstract) name of the service, since a role may provide more than one (standard) service. Used for routing.</p> <p>Multiplicity:</p>
<p>repeatCounter:RepeatCounter</p> <p>Number of times this message has already been sent. Default is zero.</p> <p>Multiplicity: [0..1]</p>
<p>senderId:LengthRestrictedOrganisationId</p> <p>The organisation ID of the message's sender.</p> <p>Multiplicity:</p>
<p>senderRole:PartnerRoleCode</p> <p>The electronic fare management (EFM) role of the sender.</p> <p>Multiplicity:</p>
<p>senderService:ServiceName</p> <p>The (abstract) name of the service, since a role may provide more than one (standard) service.</p> <p>Multiplicity:</p>

8.5.7 ionRoutingHeader

Element for usage inside the WSDLs. Defines a routing header for requests in the binding.

8.5.8 LengthRestrictedOrganisationId

Restricts the length of any used organisation ID to 255 digits.

8.5.9 OptionalRoutingInfo

Allows optional routing information for local ION access points.

8.5.10 DeliveredToEnum

Enumeration indicating to which location the message has been sent.

Enumeration values
Value: CRE Description: Message was delivered to CRE (temporarily stored or refused there)
Value: OTHER Description: Message was delivered to another location (JSB, etc.)
Value: RECIPIENT Description: Message was delivered to the recipient (accepted or refused)

8.6 ion-communication-event-enums

8.6.1 IonCommunicationErrorCode

Error Codes of a technical delivery rejection from the central routing engine (CRE) for asynchronous messages. For semantic evaluation see IonCommunicationErrorEnum.

8.6.2 IonCommunicationErrorEnum

Status Codes of a technical delivery rejection from the central routing engine (CRE) or receiving system for asynchronous messages.

Enumeration values
Value: E_IONC_CRE_COULD_NOT_STORE_MESSAGE Description: The message could not be queued for later forwarding.
Value: E_IONC_HEADER_TO_TLS_CERTIFICATE_MISMATCH

Description: The sender ID transferred in the SOAP header does not match the sender ID of the TLS certificate.

Value: E_IONC_INVALID_SOAP_HEADER

Description: Invalid Message. Sender ID or routing information could not be extracted from the SOAP header.

Value: E_IONC_NO_TARGET_ADDRESS_FOUND_FOR_ROUTING_PARAMETERS

Description: No target address found for routing parameters.

Value: E_IONC_RECEIVER_NOT_REACHABLE

Description: A connection to the receiver could not be established.

Value: E_IONC_RECEIVER_RESPONSE_TIMEOUT

Description: The receiver does not respond in the specified period.

Value: E_IONC_UNKNOWN_OPERATION

Description: The operation name in the ION message header is unknown in the CRE.

Value: E_IONC_UNKNOWN_OR_UNAUTHORISED_RECEIVER

Description: The receiver organisation ID is deactivated or unknown.

Value: E_IONC_UNKNOWN_OR_UNAUTHORISED_SENDER

Description: The sender organisation ID is deactivated or unknown.

8.7 ion-communication-event-types

8.7.1 E_IONC_CRE_COULD_NOT_STORE_MESSAGE

The message could not be queued for later forwarding.

8.7.2 E_IONC_HEADER_TO_TLS_CERTIFICATE_MISMATCH

The sender ID transferred in the SOAP header does not match the sender ID of the TLS certificate.

8.7.3 E_IONC_INVALID_SOAP_HEADER

Invalid Message. Sender ID or routing information could not be extracted from the SOAP header.

8.7.4 E_IONC_NO_TARGET_ADDRESS_FOUND_FOR_ROUTING_PARAMETERS

No target address found for routing parameters.

8.7.5 E_IONC_RECEIVER_NOT_REACHABLE

A connection to the receiver could not be established.

8.7.6 E_IONC_RECEIVER_RESPONSE_TIMEOUT

The receiver does not respond in the specified period.

8.7.7 E_IONC_UNKNOWN_OPERATION

The operation name in the ION message header is unknown in the CRE.

8.7.8 E_IONC_UNKNOWN_OR_UNAUTHORISED_RECEIVER

The receiver organisation ID is deactivated or unknown.

8.7.9 E_IONC_UNKNOWN_OR_UNAUTHORISED_SENDER

The sender organisation ID is deactivated or unknown.

8.8 cre-messaging

8.8.1 setServiceAvailable

Input element: Sets the service (service+orgId+role) as available.

8.8.2 SetServiceAvailable

Input type: Sets the service (service+orgId+role) as available.

8.8.3 setServiceAvailableException

Sets the service (service+orgId+role) as available.

8.8.4 setServiceAvailableResponse

Output element: Sets the service (service+orgId+role) as available.

8.8.5 SetServiceAvailableResponse

Output type: Sets the service (service+orgId+role) as available.

8.8.6 setServiceUnavailable

Input element: Sets the service (service+orgId+role) as unavailable.

8.8.7 SetServiceUnavailable

Input type: Sets the service (service+orgId+role) as unavailable.

8.8.8 setServiceUnavailableException

Sets the service (service+orgId+role) as unavailable.

8.8.9 setServiceUnavailableResponse

Output element: Sets the service (service+orgId+role) as unavailable.

8.8.10 SetServiceUnavailableResponse

Output type: Sets the service (service+orgId+role) as unavailable.

8.9 cre-event-enums

8.9.1 CreErrorEnum

Enumeration of errors which might occur while enabling or disabling services.

Enumeration values
Value: E_CRE_SERVICE_NOT_CONFIGURED Description: Service is not configured. The caller has sent a service name that is not configured in the CRE.

8.10 cre-event-types

8.10.1 E_CRE_SERVICE_NOT_CONFIGURED

Service is not configured. The caller has sent a service name that is not configured in the CRE.

Appendix: List of References

List of Specification References

1 [1] etiCORE Interfaces: ASN.1, XSD and WSDL

See <https://modell.eticket-deutschland.de/>

2 [2] etiCORE ION Adapter Specification

Specification of the ION adapter to connect system components to the ION

3 [3] CRE Specification: Message Store and Forward

Specification of the CRE that covers the message store & forward mechanism

4 [4] Documentation Registrar System for Service Configuration

Part of the ESH that allows the system configuration for ION services

5 [5] SLAs for Message Transfer

- Maximum time span for processing a message (e.g. single or multiple notifications) through the whole processing chain: 7 days in ION plus 1 day in terminal
- Chunks for multi-messages: 10,000 messages in the list.
- Check if the message time stamp is still within in the maximal allowed delivery time span. The time span is 300 seconds.

List of Webservice Security References

6 [6] XML Encryption Standard

<https://www.w3.org/TR/xmlenc-core/>

7 [7] XML Signature Standard

<https://www.w3.org/TR/xmldsig-core/>

8 [8] WS-Security 1.1

WS-Security 1.1: <https://www.oasis-open.org/specs/index.php#wssv1.1>

9 [9] WS-SecurityPolicy v1.2

WS-SecurityPolicy v1.2: <https://www.oasis-open.org/specs/index.php/wssecpolv1.2>

10 [10] RFC 3268

RFC 3268: <http://www.ietf.org/rfc/rfc3268.txt>

11 [11] RFC 5280

RFC 5280: <http://www.ietf.org/rfc/rfc5280.txt>

12 [12] RFC 5246

RFC for TLS 1.2, see <https://www.ietf.org/rfc/rfc5246.txt>

