
MODULE *E2AP*

The *E2AP* module provides a formal specification of the *E2AP* protocol. The spec defines the client and server interfaces for *E2AP* and provides helpers for managing and operating on connections.

LOCAL INSTANCE *Naturals*

LOCAL INSTANCE *Sequences*

LOCAL INSTANCE *FiniteSets*

LOCAL INSTANCE *TLC*

CONSTANT *Nil*

VARIABLE *conns*

The *E2AP* protocol is implemented on *SCTP*

LOCAL *SCTP* \triangleq INSTANCE *SCTP*

vars \triangleq $\langle \textit{conns} \rangle$

MODULE *Cause*

The *Messages* module defines predicates for receiving, sending, and verifying all the messages supported by *E2AP*.

MODULE *Misc*

CONSTANTS

Unspecified,
ControlProcessingOverload,
HardwareFailure,
OMIntervention

All \triangleq
 $\{ \textit{Unspecified}$,
 $\textit{ControlProcessingOverload}$,
 $\textit{HardwareFailure}$,
 $\textit{OMIntervention} \}$

ASSUME $\forall c \in \textit{All} : c \in \text{STRING}$

$\textit{IsUnspecified}(m) \triangleq m.\textit{cause} = \textit{Unspecified}$
 $\textit{IsControlProcessingOverload}(m) \triangleq m.\textit{cause} = \textit{ControlProcessingOverload}$
 $\textit{IsHardwareFailure}(m) \triangleq m.\textit{cause} = \textit{HardwareFailure}$
 $\textit{IsOMIntervention}(m) \triangleq m.\textit{cause} = \textit{OMIntervention}$

Misc \triangleq INSTANCE *Misc* WITH
 $\textit{Unspecified} \leftarrow \text{"Unspecified"},$

ControlProcessingOverload \leftarrow "ControlProcessingOverload",
HardwareFailure \leftarrow "HardwareFailure",
OMIntervention \leftarrow "OMIntervention"

MODULE *Protocol*

CONSTANTS

Unspecified,
TransferSyntaxError,
AbstractSyntaxErrorReject,
AbstractSyntaxErrorIgnoreAndNotify,
MessageNotCompatibleWithReceiverState,
SemanticError,
AbstractSyntaxErrorFalselyConstructedMessage

All \triangleq

{ *Unspecified*,
TransferSyntaxError,
AbstractSyntaxErrorReject,
AbstractSyntaxErrorIgnoreAndNotify,
MessageNotCompatibleWithReceiverState,
SemanticError,
AbstractSyntaxErrorFalselyConstructedMessage }

ASSUME $\forall c \in All : c \in \text{STRING}$

IsUnspecified(*m*) $\triangleq m.cause = Unspecified$
IsTransferSyntaxError(*m*) $\triangleq m.cause = TransferSyntaxError$
IsAbstractSyntaxErrorReject(*m*) $\triangleq m.cause = AbstractSyntaxErrorReject$
IsAbstractSyntaxErrorIgnoreAndNotify(*m*) $\triangleq m.cause = AbstractSyntaxErrorIgnoreAndNotify$
IsMessageNotCompatibleWithReceiverState(*m*) $\triangleq m.cause = MessageNotCompatibleWithReceiverState$
IsSemanticError(*m*) $\triangleq m.cause = SemanticError$
IsAbstractSyntaxErrorFalselyConstructedMessage(*m*) $\triangleq m.cause = AbstractSyntaxErrorFalselyConstructedMessage$

Protocol \triangleq INSTANCE *Protocol* WITH

Unspecified \leftarrow "Unspecified",
TransferSyntaxError \leftarrow "TransferSyntaxError",
AbstractSyntaxErrorReject \leftarrow "AbstractSyntaxErrorReject",
AbstractSyntaxErrorIgnoreAndNotify \leftarrow "AbstractSyntaxErrorIgnoreAndNotify",
MessageNotCompatibleWithReceiverState \leftarrow "MessageNotCompatibleWithReceiverState",
SemanticError \leftarrow "SemanticError",
AbstractSyntaxErrorFalselyConstructedMessage \leftarrow "AbstractSyntaxErrorFalselyConstructedMessage"

MODULE *RIC*

CONSTANTS

Unspecified,
RANFunctionIDInvalid,
ActionNotSupported,
ExcessiveActions,
DuplicateAction,
DuplicateEvent,
FunctionResourceLimit,
RequestIDUnknown,
InconsistentActionSubsequentActionSequence,
ControlMessageInvalid,
CallProcessIDInvalid

$All \triangleq$
 $\{$ *Unspecified*,
RANFunctionIDInvalid,
ActionNotSupported,
ExcessiveActions,
DuplicateAction,
DuplicateEvent,
FunctionResourceLimit,
RequestIDUnknown,
InconsistentActionSubsequentActionSequence,
ControlMessageInvalid,
CallProcessIDInvalid $\}$

ASSUME $\forall c \in All : c \in \text{STRING}$

$IsUnspecified(m) \triangleq m.cause = Unspecified$
 $IsRANFunctionIDInvalid(m) \triangleq m.cause = RANFunctionIDInvalid$
 $IsActionNotSupported(m) \triangleq m.cause = ActionNotSupported$
 $IsExcessiveActions(m) \triangleq m.cause = ExcessiveActions$
 $IsDuplicateAction(m) \triangleq m.cause = DuplicateAction$
 $IsDuplicateEvent(m) \triangleq m.cause = DuplicateEvent$
 $IsFunctionResourceLimit(m) \triangleq m.cause = FunctionResourceLimit$
 $IsRequestIDUnknown(m) \triangleq m.cause = RequestIDUnknown$
 $IsInconsistentActionSubsequentActionSequence(m) \triangleq m.cause = InconsistentActionSubsequentActionSequence$
 $IsControlMessageInvalid(m) \triangleq m.cause = ControlMessageInvalid$
 $IsCallProcessIDInvalid(m) \triangleq m.cause = CallProcessIDInvalid$

$RIC \triangleq$ INSTANCE RIC WITH
 $Unspecified \leftarrow \text{"Unspecified"}$,
 $RANFunctionIDInvalid \leftarrow \text{"RANFunctionIDInvalid"}$,
 $ActionNotSupported \leftarrow \text{"ActionNotSupported"}$,
 $ExcessiveActions \leftarrow \text{"ExcessiveActions"}$,

DuplicateAction \leftarrow "DuplicateAction",
DuplicateEvent \leftarrow "DuplicateEvent",
FunctionResourceLimit \leftarrow "FunctionResourceLimit",
RequestIDUnknown \leftarrow "RequestIDUnknown",
InconsistentActionSubsequentActionSequence \leftarrow "InconsistentActionSubsequentActionSequence",
ControlMessageInvalid \leftarrow "ControlMessageInvalid",
CallProcessIDInvalid \leftarrow "CallProcessIDInvalid"

MODULE *RICService*

CONSTANTS

Unspecified,
FunctionNotRequired,
ExcessiveFunctions,
RICResourceLimit

All \triangleq
 { *Unspecified*,
FunctionNotRequired,
ExcessiveFunctions,
RICResourceLimit }

ASSUME $\forall c \in All : c \in \text{STRING}$

IsUnspecified(*m*) \triangleq *m.cause* = *Unspecified*
IsFunctionNotRequired(*m*) \triangleq *m.cause* = *FunctionNotRequired*
IsExcessiveFunctions(*m*) \triangleq *m.cause* = *ExcessiveFunctions*
IsRICResourceLimit(*m*) \triangleq *m.cause* = *RICResourceLimit*

RICService \triangleq INSTANCE *RICService* WITH

Unspecified \leftarrow "Unspecified",
FunctionNotRequired \leftarrow "FunctionNotRequired",
ExcessiveFunctions \leftarrow "ExcessiveFunctions",
RICResourceLimit \leftarrow "RICResourceLimit"

MODULE *Transport*

CONSTANTS

Unspecified,
TransportResourceUnavailable

All \triangleq
 { *Unspecified*,
TransportResourceUnavailable }

ASSUME $\forall c \in All : c \in \text{STRING}$

$$IsUnspecified(m) \triangleq m.cause = Unspecified$$

$$IsTransportResourceUnavailable(m) \triangleq m.cause = TransportResourceUnavailable$$

$Transport \triangleq$ INSTANCE $Transport$ WITH
 $Unspecified \leftarrow$ "Unspecified",
 $TransportResourceUnavailable \leftarrow$ "TransportResourceUnavailable"

$All \triangleq Misc!All \cup Protocol!All \cup RIC!All \cup RICService!All \cup Transport!All$

$IsCause(c) \triangleq c \in All$

This section defines predicates for identifying *E2AP* message types on the network.

The *Cause* module provides failure causes

$Cause \triangleq$ INSTANCE $Cause$

MODULE *Messages*

The *Messages* module defines predicates for receiving, sending, and verifying all the messages supported by *E2AP*.

Message type constants

CONSTANTS

$E2SetupRequest$,
 $E2SetupResponse$,
 $E2SetupFailure$

CONSTANTS

$RICServiceUpdate$,
 $RICServiceUpdateAcknowledge$,
 $RICServiceUpdateFailure$

CONSTANTS

$ResetRequest$,
 $ResetResponse$

CONSTANTS

$RICSubscriptionRequest$,
 $RICSubscriptionResponse$,
 $RICSubscriptionFailure$

CONSTANTS

$RICSubscriptionDeleteRequest$,
 $RICSubscriptionDeleteResponse$,
 $RICSubscriptionDeleteFailure$

CONSTANTS

$RICIndication$

CONSTANTS

$RICControlRequest$,

RICControlResponse,
RICControlFailure

CONSTANTS

E2ConnectionUpdate,
E2ConnectionUpdateAcknowledge,
E2ConnectionUpdateFailure

CONSTANTS

E2NodeConfigurationUpdate,
E2NodeConfigurationUpdateAcknowledge,
E2NodeConfigurationUpdateFailure

LOCAL *messageTypes* \triangleq

{*E2SetupRequest*,
E2SetupResponse,
E2SetupFailure,
RICServiceUpdate,
RICServiceUpdateAcknowledge,
RICServiceUpdateFailure,
ResetRequest,
ResetResponse,
RICSubscriptionRequest,
RICSubscriptionResponse,
RICSubscriptionFailure,
RICSubscriptionDeleteRequest,
RICSubscriptionDeleteResponse,
RICSubscriptionDeleteFailure,
RICControlRequest,
RICControlResponse,
RICControlFailure,
RICServiceUpdate,
E2ConnectionUpdate,
E2ConnectionUpdateAcknowledge,
E2ConnectionUpdateFailure,
E2NodeConfigurationUpdate,
E2NodeConfigurationUpdateAcknowledge,
E2NodeConfigurationUpdateFailure}

Message types should be defined as strings to simplify debugging

ASSUME $\forall m \in \text{messageTypes} : m \in \text{STRING}$

This section defines predicates for identifying *E2AP* message types on the network.

IsE2SetupRequest(*msg*) \triangleq *msg.type* = *E2SetupRequest*

IsE2SetupResponse(*msg*) \triangleq *msg.type* = *E2SetupResponse*

$IsE2SetupFailure(msg) \triangleq msg.type = E2SetupFailure$
 $IsRICServiceUpdate(msg) \triangleq msg.type = RICServiceUpdate$
 $IsRICServiceUpdateAcknowledge(msg) \triangleq msg.type = RICServiceUpdateAcknowledge$
 $IsRICServiceUpdateFailure(msg) \triangleq msg.type = RICServiceUpdateFailure$
 $IsResetRequest(msg) \triangleq msg.type = ResetRequest$
 $IsResetResponse(msg) \triangleq msg.type = ResetResponse$
 $IsRICSubscriptionRequest(msg) \triangleq msg.type = RICSubscriptionRequest$
 $IsRICSubscriptionResponse(msg) \triangleq msg.type = RICSubscriptionResponse$
 $IsRICSubscriptionFailure(msg) \triangleq msg.type = RICSubscriptionFailure$
 $IsRICSubscriptionDeleteRequest(msg) \triangleq msg.type = RICSubscriptionDeleteRequest$
 $IsRICSubscriptionDeleteResponse(msg) \triangleq msg.type = RICSubscriptionDeleteResponse$
 $IsRICSubscriptionDeleteFailure(msg) \triangleq msg.type = RICSubscriptionDeleteFailure$
 $IsRICIndication(msg) \triangleq msg.type = RICIndication$
 $IsRICControlRequest(msg) \triangleq msg.type = RICControlRequest$
 $IsRICControlResponse(msg) \triangleq msg.type = RICControlResponse$
 $IsRICControlFailure(msg) \triangleq msg.type = RICControlFailure$
 $IsE2ConnectionUpdate(msg) \triangleq msg.type = E2ConnectionUpdate$
 $IsE2ConnectionUpdateAcknowledge(msg) \triangleq msg.type = E2ConnectionUpdateAcknowledge$
 $IsE2ConnectionUpdateFailure(msg) \triangleq msg.type = E2ConnectionUpdateFailure$
 $IsE2NodeConfigurationUpdate(msg) \triangleq msg.type = E2NodeConfigurationUpdate$
 $IsE2NodeConfigurationUpdateAcknowledge(msg) \triangleq msg.type = E2NodeConfigurationUpdateAcknowledge$
 $IsE2NodeConfigurationUpdateFailure(msg) \triangleq msg.type = E2NodeConfigurationUpdateFailure$

This section defines predicates for validating *E2AP* message contents. The predicates provide precise documentation on the *E2AP* message format and are used within the spec to verify that steps adhere to the *E2AP* protocol specification.

LOCAL $ValidE2SetupRequest(msg) \triangleq$
 $\quad \wedge \quad \wedge \text{"transactionId"} \in \text{DOMAIN } msg$
 $\quad \quad \wedge msg[\text{"transactionId"}] \in Nat$
 $\quad \wedge \quad \wedge \text{"globalE2NodeId"} \in \text{DOMAIN } msg$

$$\begin{aligned}
& \wedge \text{msg}[\text{"globalE2NodeId"}] \in \text{Nat} \\
\text{LOCAL } \text{ValidE2SetupResponse}(\text{msg}) & \triangleq \\
& \wedge \wedge \text{"transactionId"} \in \text{DOMAIN } \text{msg} \\
& \wedge \text{msg}[\text{"transactionId"}] \in \text{Nat} \\
& \wedge \wedge \text{"globalRicId"} \in \text{DOMAIN } \text{msg} \\
& \wedge \text{msg}[\text{"globalRicId"}] \in \text{Nat} \\
\text{LOCAL } \text{ValidE2SetupFailure}(\text{msg}) & \triangleq \\
& \wedge \wedge \text{"transactionId"} \in \text{DOMAIN } \text{msg} \\
& \wedge \text{msg}[\text{"transactionId"}] \in \text{Nat} \\
& \wedge \wedge \text{"cause"} \in \text{DOMAIN } \text{msg} \\
& \wedge \text{msg}[\text{"cause"}] \in \text{Cause!All} \\
\text{LOCAL } \text{ValidRICServiceUpdate}(\text{msg}) & \triangleq \\
& \wedge \wedge \text{"transactionId"} \in \text{DOMAIN } \text{msg} \\
& \wedge \text{msg}[\text{"transactionId"}] \in \text{Nat} \\
\text{LOCAL } \text{ValidRICServiceUpdateAcknowledge}(\text{msg}) & \triangleq \\
& \wedge \wedge \text{"transactionId"} \in \text{DOMAIN } \text{msg} \\
& \wedge \text{msg}[\text{"transactionId"}] \in \text{Nat} \\
\text{LOCAL } \text{ValidRICServiceUpdateFailure}(\text{msg}) & \triangleq \\
& \wedge \wedge \text{"transactionId"} \in \text{DOMAIN } \text{msg} \\
& \wedge \text{msg}[\text{"transactionId"}] \in \text{Nat} \\
& \wedge \wedge \text{"cause"} \in \text{DOMAIN } \text{msg} \\
& \wedge \text{msg}[\text{"cause"}] \in \text{Cause!All} \\
\text{LOCAL } \text{ValidResetRequest}(\text{msg}) & \triangleq \\
& \wedge \wedge \text{"transactionId"} \in \text{DOMAIN } \text{msg} \\
& \wedge \text{msg}[\text{"transactionId"}] \in \text{Nat} \\
\text{LOCAL } \text{ValidResetResponse}(\text{msg}) & \triangleq \\
& \wedge \wedge \text{"transactionId"} \in \text{DOMAIN } \text{msg} \\
& \wedge \text{msg}[\text{"transactionId"}] \in \text{Nat} \\
\text{LOCAL } \text{ValidE2ConnectionUpdate}(\text{msg}) & \triangleq \\
& \wedge \wedge \text{"transactionId"} \in \text{DOMAIN } \text{msg} \\
& \wedge \text{msg}[\text{"transactionId"}] \in \text{Nat} \\
& \wedge \wedge \text{"add"} \in \text{DOMAIN } \text{msg} \Rightarrow \\
& \quad \wedge \text{IsFiniteSet}(\text{msg}[\text{"add"}]) \\
& \quad \wedge \forall a \in \text{msg}[\text{"add"}] : a \in \text{STRING} \\
& \wedge \wedge \text{"update"} \in \text{DOMAIN } \text{msg} \Rightarrow \\
& \quad \wedge \text{IsFiniteSet}(\text{msg}[\text{"update"}]) \\
& \quad \wedge \forall a \in \text{msg}[\text{"update"}] : a \in \text{STRING} \\
& \wedge \wedge \text{"remove"} \in \text{DOMAIN } \text{msg} \Rightarrow \\
& \quad \wedge \text{IsFiniteSet}(\text{msg}[\text{"remsgove"}])
\end{aligned}$$

$$\wedge \forall a \in msg["remove"] : a \in \text{STRING}$$

LOCAL $ValidE2ConnectionUpdateAcknowledge(msg) \triangleq$

$$\begin{aligned} &\wedge \wedge \text{"transactionId"} \in \text{DOMAIN } msg \\ &\wedge msg["transactionId"] \in \text{Nat} \\ &\wedge \wedge \text{"succeeded"} \in \text{DOMAIN } msg \Rightarrow \\ &\quad \wedge IsFiniteSet(msg["succeeded"]) \\ &\quad \wedge \forall a \in msg["succeeded"] : a \in \text{STRING} \\ &\wedge \wedge \text{"failed"} \in \text{DOMAIN } msg \Rightarrow \\ &\quad \wedge IsFiniteSet(msg["failed"]) \\ &\quad \wedge \forall a \in msg["failed"] : a \in \text{STRING} \end{aligned}$$

LOCAL $ValidE2ConnectionUpdateFailure(msg) \triangleq$

$$\begin{aligned} &\wedge \wedge \text{"transactionId"} \in \text{DOMAIN } msg \\ &\wedge msg["transactionId"] \in \text{Nat} \\ &\wedge \wedge \text{"cause"} \in \text{DOMAIN } msg \\ &\wedge msg["cause"] \in \text{Cause!All} \end{aligned}$$

LOCAL $ValidE2NodeConfigurationUpdate(msg) \triangleq$

$$\begin{aligned} &\wedge \wedge \text{"transactionId"} \in \text{DOMAIN } msg \\ &\wedge msg["transactionId"] \in \text{Nat} \\ &\wedge \wedge \text{"globalE2NodeId"} \in \text{DOMAIN } msg \\ &\wedge msg["globalE2NodeId"] \in \text{Nat} \\ &\wedge \wedge \text{"add"} \in \text{DOMAIN } msg \Rightarrow \\ &\quad \wedge IsFiniteSet(msg["add"]) \\ &\wedge \wedge \text{"update"} \in \text{DOMAIN } msg \Rightarrow \\ &\quad \wedge IsFiniteSet(msg["update"]) \\ &\wedge \wedge \text{"remove"} \in \text{DOMAIN } msg \Rightarrow \\ &\quad \wedge IsFiniteSet(msg["remove"]) \end{aligned}$$

LOCAL $ValidE2NodeConfigurationUpdateAcknowledge(msg) \triangleq$

$$\begin{aligned} &\wedge \wedge \text{"transactionId"} \in \text{DOMAIN } msg \\ &\wedge msg["transactionId"] \in \text{Nat} \\ &\wedge \wedge \text{"add"} \in \text{DOMAIN } msg \Rightarrow \\ &\quad \wedge IsFiniteSet(msg["add"]) \\ &\wedge \wedge \text{"update"} \in \text{DOMAIN } msg \Rightarrow \\ &\quad \wedge IsFiniteSet(msg["update"]) \\ &\wedge \wedge \text{"remove"} \in \text{DOMAIN } msg \Rightarrow \\ &\quad \wedge IsFiniteSet(msg["remove"]) \end{aligned}$$

LOCAL $ValidE2NodeConfigurationUpdateFailure(msg) \triangleq$

$$\begin{aligned} &\wedge \wedge \text{"transactionId"} \in \text{DOMAIN } msg \\ &\wedge msg["transactionId"] \in \text{Nat} \\ &\wedge \wedge \text{"cause"} \in \text{DOMAIN } msg \\ &\wedge msg["cause"] \in \text{Cause!All} \end{aligned}$$

LOCAL $ValidRICSubscriptionRequest(msg) \triangleq$

$$\begin{aligned}
& \wedge \quad \wedge \text{ "requestId" } \in \text{DOMAIN } msg \\
& \quad \wedge msg[\text{ "requestId" }] \in Nat \\
\text{LOCAL } & \text{ValidRICSubscriptionResponse}(msg) \triangleq \\
& \quad \wedge \quad \wedge \text{ "requestId" } \in \text{DOMAIN } msg \\
& \quad \quad \wedge msg[\text{ "requestId" }] \in Nat \\
\text{LOCAL } & \text{ValidRICSubscriptionFailure}(msg) \triangleq \\
& \quad \wedge \quad \wedge \text{ "requestId" } \in \text{DOMAIN } msg \\
& \quad \quad \wedge msg[\text{ "requestId" }] \in Nat \\
& \quad \wedge \quad \wedge \text{ "cause" } \in \text{DOMAIN } msg \\
& \quad \quad \wedge msg[\text{ "cause" }] \in Cause!All \\
\text{LOCAL } & \text{ValidRICSubscriptionDeleteRequest}(msg) \triangleq \\
& \quad \wedge \quad \wedge \text{ "requestId" } \in \text{DOMAIN } msg \\
& \quad \quad \wedge msg[\text{ "requestId" }] \in Nat \\
\text{LOCAL } & \text{ValidRICSubscriptionDeleteResponse}(msg) \triangleq \\
& \quad \wedge \quad \wedge \text{ "requestId" } \in \text{DOMAIN } msg \\
& \quad \quad \wedge msg[\text{ "requestId" }] \in Nat \\
\text{LOCAL } & \text{ValidRICSubscriptionDeleteFailure}(msg) \triangleq \\
& \quad \wedge \quad \wedge \text{ "requestId" } \in \text{DOMAIN } msg \\
& \quad \quad \wedge msg[\text{ "requestId" }] \in Nat \\
& \quad \wedge \quad \wedge \text{ "cause" } \in \text{DOMAIN } msg \\
& \quad \quad \wedge msg[\text{ "cause" }] \in Cause!All \\
\text{LOCAL } & \text{ValidRICIndication}(msg) \triangleq \\
& \quad \wedge \quad \wedge \text{ "requestId" } \in \text{DOMAIN } msg \\
& \quad \quad \wedge msg[\text{ "requestId" }] \in Nat \\
\text{LOCAL } & \text{ValidRICControlRequest}(msg) \triangleq \\
& \quad \wedge \quad \wedge \text{ "requestId" } \in \text{DOMAIN } msg \\
& \quad \quad \wedge msg[\text{ "requestId" }] \in Nat \\
\text{LOCAL } & \text{ValidRICControlAcknowledge}(msg) \triangleq \\
& \quad \wedge \quad \wedge \text{ "requestId" } \in \text{DOMAIN } msg \\
& \quad \quad \wedge msg[\text{ "requestId" }] \in Nat \\
\text{LOCAL } & \text{ValidRICControlFailure}(msg) \triangleq \\
& \quad \wedge \quad \wedge \text{ "requestId" } \in \text{DOMAIN } msg \\
& \quad \quad \wedge msg[\text{ "requestId" }] \in Nat \\
& \quad \wedge \quad \wedge \text{ "cause" } \in \text{DOMAIN } msg \\
& \quad \quad \wedge msg[\text{ "cause" }] \in Cause!All
\end{aligned}$$

This section defines operators for constructing *E2AP* messages.

```

LOCAL SetType(msg, type)  $\triangleq$  [msg EXCEPT !type = type]

LOCAL SetFailureCause(msg, cause)  $\triangleq$  [msg EXCEPT !cause = cause]

WithE2SetupRequest(msg)  $\triangleq$ 
  IF Assert(ValidE2SetupRequest(msg), "Invalid E2SetupRequest")
  THEN SetType(msg, E2SetupRequest)
  ELSE Nil

WithE2SetupResponse(msg)  $\triangleq$ 
  IF Assert(ValidE2SetupResponse(msg), "Invalid E2SetupResponse")
  THEN SetType(msg, E2SetupResponse)
  ELSE Nil

WithE2SetupFailure(msg, cause)  $\triangleq$ 
  IF Assert(ValidE2SetupFailure(msg), "Invalid E2SetupFailure")
  THEN SetType(msg, SetFailureCause(E2SetupFailure, cause))
  ELSE Nil

WithRICServiceUpdate(msg)  $\triangleq$ 
  IF Assert(ValidRICServiceUpdate(msg), "Invalid RICServiceUpdate")
  THEN SetType(msg, RICServiceUpdate)
  ELSE Nil

WithRICServiceUpdateAcknowledge(msg)  $\triangleq$ 
  IF Assert(ValidRICServiceUpdateAcknowledge(msg), "Invalid RICServiceUpdateAcknowledge")
  THEN SetType(msg, RICServiceUpdateAcknowledge)
  ELSE Nil

WithRICServiceUpdateFailure(msg, cause)  $\triangleq$ 
  IF Assert(ValidRICServiceUpdateFailure(msg), "Invalid RICServiceUpdateFailure")
  THEN SetType(msg, SetFailureCause(RICServiceUpdateFailure, cause))
  ELSE Nil

WithResetRequest(msg)  $\triangleq$ 
  IF Assert(ValidResetRequest(msg), "Invalid ResetRequest")
  THEN SetType(msg, ResetRequest)
  ELSE Nil

WithResetResponse(msg)  $\triangleq$ 
  IF Assert(ValidResetResponse(msg), "Invalid ResetResponse")
  THEN SetType(msg, ResetResponse)
  ELSE Nil

WithRICSubscriptionRequest(msg)  $\triangleq$ 
  IF Assert(ValidRICSubscriptionRequest(msg), "Invalid RICSubscriptionRequest")
  THEN SetType(msg, RICSubscriptionRequest)
  ELSE Nil

```

$WithRICSubscriptionResponse(msg) \triangleq$
 IF $Assert(ValidRICSubscriptionResponse(msg), \text{"Invalid RICSubscriptionResponse"})$
 THEN $SetType(msg, RICSubscriptionResponse)$
 ELSE Nil

$WithRICSubscriptionFailure(msg, cause) \triangleq$
 IF $Assert(ValidRICSubscriptionFailure(msg), \text{"Invalid RICSubscriptionFailure"})$
 THEN $SetType(msg, SetFailureCause(RICSubscriptionFailure, cause))$
 ELSE Nil

$WithRICSubscriptionDeleteRequest(msg) \triangleq$
 IF $Assert(ValidRICSubscriptionDeleteRequest(msg), \text{"Invalid RICSubscriptionDeleteRequest"})$
 THEN $SetType(msg, RICSubscriptionDeleteRequest)$
 ELSE Nil

$WithRICSubscriptionDeleteResponse(msg) \triangleq$
 IF $Assert(ValidRICSubscriptionDeleteResponse(msg), \text{"Invalid RICSubscriptionDeleteResponse"})$
 THEN $SetType(msg, RICSubscriptionDeleteResponse)$
 ELSE Nil

$WithRICSubscriptionDeleteFailure(msg, cause) \triangleq$
 IF $Assert(ValidRICSubscriptionDeleteFailure(msg), \text{"Invalid RICSubscriptionDeleteFailure"})$
 THEN $SetType(msg, SetFailureCause(RICSubscriptionDeleteFailure, cause))$
 ELSE Nil

$WithRICIndication(msg) \triangleq$
 IF $Assert(ValidRICIndication(msg), \text{"Invalid RICIndication"})$
 THEN $SetType(msg, RICIndication)$
 ELSE Nil

$WithRICControlRequest(msg) \triangleq$
 IF $Assert(ValidRICControlRequest(msg), \text{"Invalid RICControlRequest"})$
 THEN $SetType(msg, RICControlRequest)$
 ELSE Nil

$WithRICControlAcknowledge(msg) \triangleq$
 IF $Assert(ValidRICControlAcknowledge(msg), \text{"Invalid RICControlAcknowledge"})$
 THEN $SetType(msg, RICControlResponse)$
 ELSE Nil

$WithRICControlFailure(msg, cause) \triangleq$
 IF $Assert(ValidRICControlFailure(msg), \text{"Invalid RICControlFailure"})$
 THEN $SetType(msg, SetFailureCause(RICControlFailure, cause))$
 ELSE Nil

$WithE2ConnectionUpdate(msg) \triangleq$
 IF $Assert(ValidE2ConnectionUpdate(msg), \text{"Invalid E2ConnectionUpdate"})$
 THEN $SetType(msg, E2ConnectionUpdate)$

```

ELSE Nil

WithE2ConnectionUpdateAcknowledge(msg)  $\triangleq$ 
  IF Assert(ValidE2ConnectionUpdateAcknowledge(msg), "Invalid E2ConnectionUpdateAcknowledge")
  THEN SetType(msg, E2ConnectionUpdateAcknowledge)
  ELSE Nil

WithE2ConnectionUpdateFailure(msg, cause)  $\triangleq$ 
  IF Assert(ValidE2ConnectionUpdateFailure(msg), "Invalid E2ConnectionUpdateFailure")
  THEN SetType(msg, SetFailureCause(E2ConnectionUpdateFailure, cause))
  ELSE Nil

WithE2NodeConfigurationUpdate(msg)  $\triangleq$ 
  IF Assert(ValidE2NodeConfigurationUpdate(msg), "Invalid E2NodeConfigurationUpdate")
  THEN SetType(msg, E2NodeConfigurationUpdate)
  ELSE Nil

WithE2NodeConfigurationUpdateAcknowledge(msg)  $\triangleq$ 
  IF Assert(ValidE2NodeConfigurationUpdateAcknowledge(msg), "Invalid E2NodeConfigurationUpdateAckn")
  THEN SetType(msg, E2NodeConfigurationUpdateAcknowledge)
  ELSE Nil

WithE2NodeConfigurationUpdateFailure(msg, cause)  $\triangleq$ 
  IF Assert(ValidE2NodeConfigurationUpdateFailure(msg), "Invalid E2NodeConfigurationUpdateFailure")
  THEN SetType(msg, SetFailureCause(E2NodeConfigurationUpdateFailure, cause))
  ELSE Nil

```

The *Messages* module is instantiated locally to avoid access from outside the module.

```

LOCAL Messages  $\triangleq$  INSTANCE Messages WITH
  E2SetupRequest  $\leftarrow$  "E2SetupRequest",
  E2SetupResponse  $\leftarrow$  "E2SetupResponse",
  E2SetupFailure  $\leftarrow$  "E2SetupFailure",
  ResetRequest  $\leftarrow$  "ResetRequest",
  ResetResponse  $\leftarrow$  "ResetResponse",
  RICSubscriptionRequest  $\leftarrow$  "RICSubscriptionRequest",
  RICSubscriptionResponse  $\leftarrow$  "RICSubscriptionResponse",
  RICSubscriptionFailure  $\leftarrow$  "RICSubscriptionFailure",
  RICSubscriptionDeleteRequest  $\leftarrow$  "RICSubscriptionDeleteRequest",
  RICSubscriptionDeleteResponse  $\leftarrow$  "RICSubscriptionDeleteResponse",
  RICSubscriptionDeleteFailure  $\leftarrow$  "RICSubscriptionDeleteFailure",
  RICIndication  $\leftarrow$  "RICIndication",
  RICControlRequest  $\leftarrow$  "RICControlRequest",
  RICControlResponse  $\leftarrow$  "RICControlResponse",
  RICControlFailure  $\leftarrow$  "RICControlFailure",

```

$RICServiceUpdate \leftarrow \text{"RICServiceUpdate"},$
 $RICServiceUpdateAcknowledge \leftarrow \text{"RICServiceUpdateAcknowledge"},$
 $RICServiceUpdateFailure \leftarrow \text{"RICServiceUpdateFailure"},$
 $E2ConnectionUpdate \leftarrow \text{"E2ConnectionUpdate"},$
 $E2ConnectionUpdateAcknowledge \leftarrow \text{"E2ConnectionUpdateAcknowledge"},$
 $E2ConnectionUpdateFailure \leftarrow \text{"E2ConnectionUpdateFailure"},$
 $E2NodeConfigurationUpdate \leftarrow \text{"E2NodeConfigurationUpdate"},$
 $E2NodeConfigurationUpdateAcknowledge \leftarrow \text{"E2NodeConfigurationUpdateAcknowledge"},$
 $E2NodeConfigurationUpdateFailure \leftarrow \text{"E2NodeConfigurationUpdateFailure"}$

MODULE *Client*

The *Client* module provides operators for managing and operating on *E2AP* client connections and specifies the message types supported for the client.

CONSTANT *ID*

MODULE *Send*

This module provides message type operators for the message types that can be send by the *E2AP* client.

$E2SetupRequest(conn, msg) \triangleq$
 $\quad \wedge SCTP! Client(ID)! Send(conn, Messages! WithE2SetupResponse(msg))$
 $RICServiceUpdate(conn, msg) \triangleq$
 $\quad \wedge SCTP! Client(ID)! Send(conn, Messages! WithRICServiceUpdate(msg))$
 $ResetRequest(conn, msg) \triangleq$
 $\quad \wedge SCTP! Client(ID)! Send(conn, Messages! WithResetRequest(msg))$
 $ResetResponse(conn, msg) \triangleq$
 $\quad \wedge SCTP! Client(ID)! Send(conn, Messages! WithResetResponse(msg))$
 $RICSubscriptionResponse(conn, msg) \triangleq$
 $\quad \wedge SCTP! Client(ID)! Send(conn, Messages! WithRICSubscriptionResponse(msg))$
 $RICSubscriptionFailure(conn, msg, cause) \triangleq$
 $\quad \wedge SCTP! Client(ID)! Send(conn, Messages! WithRICSubscriptionFailure(msg, cause))$
 $RICSubscriptionDeleteResponse(conn, msg) \triangleq$
 $\quad \wedge SCTP! Client(ID)! Send(conn, Messages! WithRICSubscriptionDeleteResponse(msg))$
 $RICSubscriptionDeleteFailure(conn, msg, cause) \triangleq$
 $\quad \wedge SCTP! Client(ID)! Send(conn, Messages! WithRICSubscriptionDeleteFailure(msg, cause))$
 $RICIndication(conn, msg) \triangleq$
 $\quad \wedge SCTP! Client(ID)! Send(conn, Messages! WithRICIndication(msg))$
 $RICControlAcknowledge(conn, msg) \triangleq$
 $\quad \wedge SCTP! Client(ID)! Send(conn, Messages! WithRICControlAcknowledge(msg))$

$$\begin{aligned}
&RICControlFailure(conn, msg, cause) \triangleq \\
&\quad \wedge SCTP! Client(ID)! Send(conn, Messages! WithRICControlFailure(msg, cause)) \\
&E2ConnectionUpdate(conn, msg) \triangleq \\
&\quad \wedge SCTP! Client(ID)! Send(conn, Messages! WithE2ConnectionUpdate(msg)) \\
&E2ConnectionUpdateAcknowledge(conn, msg) \triangleq \\
&\quad \wedge SCTP! Client(ID)! Send(conn, Messages! WithE2ConnectionUpdateAcknowledge(msg)) \\
&E2NodeConfigurationUpdate(conn, msg) \triangleq \\
&\quad \wedge SCTP! Client(ID)! Send(conn, Messages! WithE2NodeConfigurationUpdate(msg)) \\
&E2NodeConfigurationUpdateAcknowledge(conn, msg) \triangleq \\
&\quad \wedge SCTP! Client(ID)! Send(conn, Messages! WithE2NodeConfigurationUpdateAcknowledge(msg))
\end{aligned}$$

Instantiate the *E2AP! Client! Requests* module

Send \triangleq INSTANCE *Send*

MODULE *Reply*

This module provides message type operators for the message types that can be send by the *E2AP* client.

$$\begin{aligned}
&ResetResponse(conn, msg) \triangleq \\
&\quad \wedge SCTP! Client(ID)! Reply(conn, Messages! WithResetResponse(msg)) \\
&RICSubscriptionResponse(conn, msg) \triangleq \\
&\quad \wedge SCTP! Client(ID)! Reply(conn, Messages! WithRICSubscriptionResponse(msg)) \\
&RICSubscriptionFailure(conn, msg, cause) \triangleq \\
&\quad \wedge SCTP! Client(ID)! Reply(conn, Messages! WithRICSubscriptionFailure(msg, cause)) \\
&RICSubscriptionDeleteResponse(conn, msg) \triangleq \\
&\quad \wedge SCTP! Client(ID)! Reply(conn, Messages! WithRICSubscriptionDeleteResponse(msg)) \\
&RICSubscriptionDeleteFailure(conn, msg, cause) \triangleq \\
&\quad \wedge SCTP! Client(ID)! Reply(conn, Messages! WithRICSubscriptionDeleteFailure(msg, cause)) \\
&RICIndication(conn, msg) \triangleq \\
&\quad \wedge SCTP! Client(ID)! Reply(conn, Messages! WithRICIndication(msg)) \\
&RICControlAcknowledge(conn, msg) \triangleq \\
&\quad \wedge SCTP! Client(ID)! Reply(conn, Messages! WithRICControlAcknowledge(msg)) \\
&RICControlFailure(conn, msg, cause) \triangleq \\
&\quad \wedge SCTP! Client(ID)! Reply(conn, Messages! WithRICControlFailure(msg, cause)) \\
&E2ConnectionUpdate(conn, msg) \triangleq \\
&\quad \wedge SCTP! Client(ID)! Reply(conn, Messages! WithE2ConnectionUpdate(msg))
\end{aligned}$$

$$\begin{aligned}
&E2ConnectionUpdateAcknowledge(conn, msg) \triangleq \\
&\quad \wedge SCTP! Client(ID)! Reply(conn, Messages! WithE2ConnectionUpdateAcknowledge(msg)) \\
&E2NodeConfigurationUpdate(conn, msg) \triangleq \\
&\quad \wedge SCTP! Client(ID)! Reply(conn, Messages! WithE2NodeConfigurationUpdate(msg)) \\
&E2NodeConfigurationUpdateAcknowledge(conn, msg) \triangleq \\
&\quad \wedge SCTP! Client(ID)! Reply(conn, Messages! WithE2NodeConfigurationUpdateAcknowledge(msg))
\end{aligned}$$

Instantiate the $E2AP! Client! Reply$ module
 $Reply \triangleq \text{INSTANCE } Reply$

MODULE *Receive*

This module provides predicates for the types of messages that can be received by an $E2AP$ client.

$$\begin{aligned}
&E2SetupResponse(conn, handler(-, -)) \triangleq \\
&\quad SCTP! Client(ID)! Handle(conn, LAMBDA c, m : \\
&\quad \quad \wedge Messages! IsE2SetupResponse(m) \\
&\quad \quad \wedge SCTP! Client(ID)! Receive(conn) \\
&\quad \quad \wedge handler(c, m)) \\
&RICServiceUpdateAcknowledge(conn, handler(-, -)) \triangleq \\
&\quad SCTP! Client(ID)! Handle(conn, LAMBDA c, m : \\
&\quad \quad \wedge Messages! IsRICServiceUpdateAcknowledge(m) \\
&\quad \quad \wedge SCTP! Client(ID)! Receive(conn) \\
&\quad \quad \wedge handler(c, m)) \\
&RICServiceUpdateFailure(conn, handler(-, -)) \triangleq \\
&\quad SCTP! Client(ID)! Handle(conn, LAMBDA c, m : \\
&\quad \quad \wedge Messages! IsRICServiceUpdateFailure(m) \\
&\quad \quad \wedge SCTP! Client(ID)! Receive(conn) \\
&\quad \quad \wedge handler(c, m)) \\
&ResetRequest(conn, handler(-, -)) \triangleq \\
&\quad SCTP! Client(ID)! Handle(conn, LAMBDA c, m : \\
&\quad \quad \wedge Messages! IsResetRequest(m) \\
&\quad \quad \wedge SCTP! Client(ID)! Receive(conn) \\
&\quad \quad \wedge handler(c, m)) \\
&ResetResponse(conn, handler(-, -)) \triangleq \\
&\quad SCTP! Client(ID)! Handle(conn, LAMBDA c, m : \\
&\quad \quad \wedge Messages! IsResetResponse(m) \\
&\quad \quad \wedge SCTP! Client(ID)! Receive(conn) \\
&\quad \quad \wedge handler(c, m))
\end{aligned}$$

$$\begin{aligned}
& \text{RICSubscriptionRequest}(\text{conn}, \text{handler}(-, -)) \triangleq \\
& \quad \text{SCTP! Client}(ID)! \text{Handle}(\text{conn}, \text{LAMBDA } c, m : \\
& \quad \quad \wedge \text{Messages! IsRICSubscriptionRequest}(m) \\
& \quad \quad \wedge \text{SCTP! Client}(ID)! \text{Receive}(\text{conn}) \\
& \quad \quad \wedge \text{handler}(c, m)) \\
& \text{RICSubscriptionDeleteRequest}(\text{conn}, \text{handler}(-, -)) \triangleq \\
& \quad \text{SCTP! Client}(ID)! \text{Handle}(\text{conn}, \text{LAMBDA } c, m : \\
& \quad \quad \wedge \text{Messages! IsRICSubscriptionDeleteRequest}(m) \\
& \quad \quad \wedge \text{SCTP! Client}(ID)! \text{Receive}(\text{conn}) \\
& \quad \quad \wedge \text{handler}(c, m)) \\
& \text{RICControlRequest}(\text{conn}, \text{handler}(-, -)) \triangleq \\
& \quad \text{SCTP! Client}(ID)! \text{Handle}(\text{conn}, \text{LAMBDA } c, m : \\
& \quad \quad \wedge \text{Messages! IsRICControlRequest}(m) \\
& \quad \quad \wedge \text{SCTP! Client}(ID)! \text{Receive}(\text{conn}) \\
& \quad \quad \wedge \text{handler}(c, m)) \\
& \text{E2ConnectionUpdate}(\text{conn}, \text{handler}(-, -)) \triangleq \\
& \quad \text{SCTP! Client}(ID)! \text{Handle}(\text{conn}, \text{LAMBDA } c, m : \\
& \quad \quad \wedge \text{Messages! IsE2ConnectionUpdate}(m) \\
& \quad \quad \wedge \text{SCTP! Client}(ID)! \text{Receive}(\text{conn}) \\
& \quad \quad \wedge \text{handler}(c, m)) \\
& \text{E2ConnectionUpdateAcknowledge}(\text{conn}, \text{handler}(-, -)) \triangleq \\
& \quad \text{SCTP! Client}(ID)! \text{Handle}(\text{conn}, \text{LAMBDA } c, m : \\
& \quad \quad \wedge \text{Messages! IsE2ConnectionUpdateAcknowledge}(m) \\
& \quad \quad \wedge \text{SCTP! Client}(ID)! \text{Receive}(\text{conn}) \\
& \quad \quad \wedge \text{handler}(c, m)) \\
& \text{E2NodeConfigurationUpdate}(\text{conn}, \text{handler}(-, -)) \triangleq \\
& \quad \text{SCTP! Client}(ID)! \text{Handle}(\text{conn}, \text{LAMBDA } c, m : \\
& \quad \quad \wedge \text{Messages! IsE2NodeConfigurationUpdate}(m) \\
& \quad \quad \wedge \text{SCTP! Client}(ID)! \text{Receive}(\text{conn}) \\
& \quad \quad \wedge \text{handler}(c, m)) \\
& \text{E2NodeConfigurationUpdateAcknowledge}(\text{conn}, \text{handler}(-, -)) \triangleq \\
& \quad \text{SCTP! Client}(ID)! \text{Handle}(\text{conn}, \text{LAMBDA } c, m : \\
& \quad \quad \wedge \text{Messages! IsE2NodeConfigurationUpdateAcknowledge}(m) \\
& \quad \quad \wedge \text{SCTP! Client}(ID)! \text{Receive}(\text{conn}) \\
& \quad \quad \wedge \text{handler}(c, m))
\end{aligned}$$

Instantiate the *E2AP! Client! Responses* module
 $\text{Handle} \triangleq \text{INSTANCE Receive}$

$\text{Connect}(dst) \triangleq \text{SCTP! Client}(ID)! \text{Connect}(dst)$

$Disconnect(conn) \triangleq SCTP!Client(ID)!Disconnect(conn)$

The set of all open *E2AP* connections
 $Connections \triangleq SCTP!Client(ID)!Connections$

Provides operators for the *E2AP* client
 $Client(ID) \triangleq \text{INSTANCE } Client$

MODULE *Server*

The *Server* module provides operators for managing and operating on *E2AP* servers and specifies the message types supported for the server.

CONSTANT *ID*

MODULE *Send*

This module provides message type operators for the message types that can be send by the *E2AP* server.

$E2SetupResponse(conn, msg) \triangleq$
 $\quad \wedge SCTP!Server(ID)!Send(conn, Messages!WithE2SetupResponse(msg))$

$RICServiceUpdateAcknowledge(conn, msg) \triangleq$
 $\quad \wedge SCTP!Server(ID)!Send(conn, Messages!WithRICServiceUpdateAcknowledge(msg))$

$RICServiceUpdateFailure(conn, msg, cause) \triangleq$
 $\quad \wedge SCTP!Server(ID)!Send(conn, Messages!WithRICServiceUpdateFailure(msg, cause))$

$ResetRequest(conn, msg) \triangleq$
 $\quad \wedge SCTP!Server(ID)!Send(conn, Messages!WithResetRequest(msg))$

$ResetResponse(conn, msg) \triangleq$
 $\quad \wedge SCTP!Server(ID)!Send(conn, Messages!WithResetResponse(msg))$

$E2ConnectionUpdate(conn, msg) \triangleq$
 $\quad \wedge SCTP!Server(ID)!Send(conn, Messages!WithE2ConnectionUpdate(msg))$

$E2ConnectionUpdateAcknowledge(conn, msg) \triangleq$
 $\quad \wedge SCTP!Server(ID)!Send(conn, Messages!WithE2ConnectionUpdateAcknowledge(msg))$

$E2NodeConfigurationUpdate(conn, msg) \triangleq$
 $\quad \wedge SCTP!Server(ID)!Send(conn, Messages!WithE2NodeConfigurationUpdate(msg))$

$E2NodeConfigurationUpdateAcknowledge(conn, msg) \triangleq$
 $\quad \wedge SCTP!Server(ID)!Send(conn, Messages!WithE2NodeConfigurationUpdateAcknowledge(msg))$

Instantiate the *E2AP!Server!Send* module
 $Send \triangleq \text{INSTANCE } Send$

MODULE *Reply*

This module provides message type operators for the message types that can be send by the *E2AP* server.

$$\begin{aligned}
 E2SetupResponse(conn, msg) &\triangleq \\
 &\wedge SCTP!Server(ID)!Reply(conn, Messages!WithE2SetupResponse(msg)) \\
 RICSERVICEUpdateAcknowledge(conn, msg) &\triangleq \\
 &\wedge SCTP!Server(ID)!Reply(conn, Messages!WithRICSERVICEUpdateAcknowledge(msg)) \\
 RICSERVICEUpdateFailure(conn, msg, cause) &\triangleq \\
 &\wedge SCTP!Server(ID)!Reply(conn, Messages!WithRICSERVICEUpdateFailure(msg, cause)) \\
 ResetRequest(conn, msg) &\triangleq \\
 &\wedge SCTP!Server(ID)!Reply(conn, Messages!WithResetRequest(msg)) \\
 ResetResponse(conn, msg) &\triangleq \\
 &\wedge SCTP!Server(ID)!Reply(conn, Messages!WithResetResponse(msg)) \\
 E2ConnectionUpdate(conn, msg) &\triangleq \\
 &\wedge SCTP!Server(ID)!Reply(conn, Messages!WithE2ConnectionUpdate(msg)) \\
 E2ConnectionUpdateAcknowledge(conn, msg) &\triangleq \\
 &\wedge SCTP!Server(ID)!Reply(conn, Messages!WithE2ConnectionUpdateAcknowledge(msg)) \\
 E2NodeConfigurationUpdate(conn, msg) &\triangleq \\
 &\wedge SCTP!Server(ID)!Reply(conn, Messages!WithE2NodeConfigurationUpdate(msg)) \\
 E2NodeConfigurationUpdateAcknowledge(conn, msg) &\triangleq \\
 &\wedge SCTP!Server(ID)!Reply(conn, Messages!WithE2NodeConfigurationUpdateAcknowledge(msg))
 \end{aligned}$$

Instantiate the *E2AP!Server!Reply* module

Reply \triangleq INSTANCE *Reply*

MODULE *Receive*

This module provides predicates for the types of messages that can be received by an *E2AP* server.

$$\begin{aligned}
 E2SetupRequest(conn, handler(-, -)) &\triangleq \\
 &SCTP!Server(ID)!Handle(conn, LAMBDA c, m : \\
 &\quad \wedge Messages!IsE2SetupRequest(m) \\
 &\quad \wedge SCTP!Server(ID)!Receive(conn) \\
 &\quad \wedge handler(c, m)) \\
 RICSERVICEUpdate(conn, handler(-, -)) &\triangleq \\
 &SCTP!Server(ID)!Handle(conn, LAMBDA c, m : \\
 &\quad \wedge Messages!IsRICSERVICEUpdate(m)
 \end{aligned}$$

$$\begin{aligned}
& \wedge SCTP!Server(ID)!Receive(conn) \\
& \wedge handler(c, m)) \\
ResetRequest(conn, handler(-, -)) & \triangleq \\
& SCTP!Server(ID)!Handle(conn, LAMBDA\ c, m : \\
& \quad \wedge Messages!IsResetRequest(m) \\
& \quad \wedge SCTP!Server(ID)!Receive(conn) \\
& \quad \wedge handler(c, m)) \\
ResetResponse(conn, handler(-, -)) & \triangleq \\
& SCTP!Server(ID)!Handle(conn, LAMBDA\ c, m : \\
& \quad \wedge Messages!IsResetResponse(m) \\
& \quad \wedge SCTP!Server(ID)!Receive(conn) \\
& \quad \wedge handler(c, m)) \\
RICSubscriptionResponse(conn, handler(-, -)) & \triangleq \\
& SCTP!Server(ID)!Handle(conn, LAMBDA\ c, m : \\
& \quad \wedge Messages!IsRICSubscriptionResponse(m) \\
& \quad \wedge SCTP!Server(ID)!Receive(conn) \\
& \quad \wedge handler(c, m)) \\
RICSubscriptionDeleteResponse(conn, handler(-, -)) & \triangleq \\
& SCTP!Server(ID)!Handle(conn, LAMBDA\ c, m : \\
& \quad \wedge Messages!IsRICSubscriptionDeleteResponse(m) \\
& \quad \wedge SCTP!Server(ID)!Receive(conn) \\
& \quad \wedge handler(c, m)) \\
RICControlResponse(conn, handler(-, -)) & \triangleq \\
& SCTP!Server(ID)!Handle(conn, LAMBDA\ c, m : \\
& \quad \wedge Messages!IsRICControlResponse(m) \\
& \quad \wedge SCTP!Server(ID)!Receive(conn) \\
& \quad \wedge handler(c, m)) \\
RICIndication(conn, handler(-, -)) & \triangleq \\
& SCTP!Server(ID)!Handle(conn, LAMBDA\ c, m : \\
& \quad \wedge Messages!IsRICIndication(m) \\
& \quad \wedge SCTP!Server(ID)!Receive(conn) \\
& \quad \wedge handler(c, m)) \\
E2ConnectionUpdate(conn, handler(-, -)) & \triangleq \\
& SCTP!Server(ID)!Handle(conn, LAMBDA\ c, m : \\
& \quad \wedge Messages!IsE2ConnectionUpdate(m) \\
& \quad \wedge SCTP!Server(ID)!Receive(conn) \\
& \quad \wedge handler(c, m)) \\
E2ConnectionUpdateAcknowledge(conn, handler(-, -)) & \triangleq \\
& SCTP!Server(ID)!Handle(conn, LAMBDA\ c, m :
\end{aligned}$$

$$\begin{aligned} E2NodeConfigurationUpdate(conn, handler(_, _)) &\triangleq \\ &SCTP!Server(ID)!Handle(conn, \text{LAMBDA } c, m : \\ &\quad \wedge Messages!IsE2NodeConfigurationUpdate(m) \\ &\quad \wedge SCTP!Server(ID)!Receive(conn) \\ &\quad \wedge handler(c, m)) \\ E2NodeConfigurationUpdateAcknowledge(conn, handler(_, _)) &\triangleq \\ &SCTP!Server(ID)!Handle(conn, \text{LAMBDA } c, m : \\ &\quad \wedge Messages!IsE2NodeConfigurationUpdateAcknowledge(m) \\ &\quad \wedge SCTP!Server(ID)!Receive(conn) \\ &\quad \wedge handler(c, m)) \end{aligned}$$

Instantiate the *E2AP!Server!Requests* module
Handle \triangleq INSTANCE *Receive*

The set of all open *E2AP* connections
Connections \triangleq *SCTP!Server(ID)!Connections*

$$\begin{aligned} & \text{Provides operators for the } E2AP \text{ server} \\ & Server(ID) \triangleq \text{INSTANCE } Server \\ & Init \triangleq SCTP!Init \\ & Next \triangleq SCTP!Next \end{aligned}$$

21