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 *servers*, *conns*

The *E2AP* protocol is implemented on *SCTP*

LOCAL *SCTP* $\triangleq$ INSTANCE *SCTP*

$vars \triangleq \langle servers, conns \rangle$

Message type constants

CONSTANTS
E2SetupRequestType,
E2SetupResponseType,
E2SetupFailureType

CONSTANTS
ResetRequestType,
ResetResponseType

CONSTANTS
RICSubscriptionRequestType,
RICSubscriptionResponseType,
RICSubscriptionFailureType

CONSTANTS
RICSubscriptionDeleteRequestType,
RICSubscriptionDeleteResponseType,
RICSubscriptionDeleteFailureType

CONSTANTS
RICControlRequestType,
RICControlResponseType,
RICControlFailureType,
RICServiceUpdateType

CONSTANTS
E2ConnectionUpdateType,
E2ConnectionUpdateAcknowledgeType,
E2ConnectionUpdateFailureType

CONSTANTS

$E2NodeConfigurationUpdateType$,
$E2NodeConfigurationUpdateAcknowledgeType$,
$E2NodeConfigurationUpdateFailureType$

LOCAL $messageTypes \triangleq$
{$E2SetupRequestType$,
$E2SetupResponseType$,
$E2SetupFailureType$,
$ResetRequestType$,
$ResetResponseType$,
$RICSubscriptionRequestType$,
$RICSubscriptionResponseType$,
$RICSubscriptionFailureType$,
$RICSubscriptionDeleteRequestType$,
$RICSubscriptionDeleteResponseType$,
$RICSubscriptionDeleteFailureType$,
$RICControlRequestType$,
$RICControlResponseType$,
$RICControlFailureType$,
$RICServiceUpdateType$,
$E2ConnectionUpdateType$,
$E2ConnectionUpdateAcknowledgeType$,
$E2ConnectionUpdateFailureType$,
$E2NodeConfigurationUpdateType$,
$E2NodeConfigurationUpdateAcknowledgeType$,
$E2NodeConfigurationUpdateFailureType$}

Message types should be defined as strings to simplify debugging
ASSUME $\forall\, m \in messageTypes : m \in$ STRING

Failure cause constants
CONSTANTS
$MiscFailureUnspecified$,
$MiscFailureControlProcessingOverload$,
$MiscFailureHardwareFailure$,
$MiscFailureOMIntervention$
CONSTANTS
$ProtocolFailureUnspecified$,
$ProtocolFailureTransferSyntaxError$,
$ProtocolFailureAbstractSyntaxErrorReject$,
$ProtocolFailureAbstractSyntaxErrorIgnoreAndNotify$,
$ProtocolFailureMessageNotCompatibleWithReceiverState$,
$ProtocolFailureSemanticError$,
$ProtocolFailureAbstractSyntaxErrorFalselyConstructedMessage$
CONSTANTS
$RICFailureUnspecified$,

RICFailureRANFunctionIDInvalid,
RICFailureActionNotSupported,
RICFailureExcessiveActions,
RICFailureDuplicateAction,
RICFailureDuplicateEvent,
RICFailureFunctionResourceLimit,
RICFailureRequestIDUnknown,
RICFailureInconsistentActionSubsequentActionSequence,
RICFailureControlMessageInvalid,
RICFailureCallProcessIDInvalid

CONSTANTS
RICServiceFailureUnspecified,
RICServiceFailureFunctionNotRequired,
RICServiceFailureExcessiveFunctions,
RICServiceFailureRICResourceLimit

CONSTANTS
TransportFailureUnspecified,
TransportFailureTransportResourceUnavailable

LOCAL *failureCauses* $\triangleq$
{*MiscFailureUnspecified*,
MiscFailureControlProcessingOverload,
MiscFailureHardwareFailure,
MiscFailureOMIntervention,
ProtocolFailureUnspecified,
ProtocolFailureTransferSyntaxError,
ProtocolFailureAbstractSyntaxErrorReject,
ProtocolFailureAbstractSyntaxErrorIgnoreAndNotify,
ProtocolFailureMessageNotCompatibleWithReceiverState,
ProtocolFailureSemanticError,
ProtocolFailureAbstractSyntaxErrorFalselyConstructedMessage,
RICFailureUnspecified,
RICFailureRANFunctionIDInvalid,
RICFailureActionNotSupported,
RICFailureExcessiveActions,
RICFailureDuplicateAction,
RICFailureDuplicateEvent,
RICFailureFunctionResourceLimit,
RICFailureRequestIDUnknown,
RICFailureInconsistentActionSubsequentActionSequence,
RICFailureControlMessageInvalid,
RICFailureCallProcessIDInvalid,
RICServiceFailureUnspecified,
RICServiceFailureFunctionNotRequired,
RICServiceFailureExcessiveFunctions,

$RICServiceFailureRICResourceLimit$,
$TransportFailureUnspecified$,
$TransportFailureTransportResourceUnavailable$}

Failure causes should be defined as strings to simplify debugging

ASSUME $\forall\, c \in failureCauses : c \in$ STRING

MODULE *Messages*

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

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

$IsE2SetupRequest(m) \triangleq m.type = E2SetupRequestType$

$IsE2SetupResponse(m) \triangleq m.type = E2SetupResponseType$

$IsE2SetupFailure(m) \triangleq m.type = E2SetupFailureType$

$IsResetRequest(m) \triangleq m.type = ResetRequestType$

$IsResetResponse(m) \triangleq m.type = ResetResponseType$

$IsRICSubscriptionRequest(m) \triangleq m.type = RICSubscriptionRequestType$

$IsRICSubscriptionResponse(m) \triangleq m.type = RICSubscriptionResponseType$

$IsRICSubscriptionFailure(m) \triangleq m.type = RICSubscriptionFailureType$

$IsRICSubscriptionDeleteRequest(m) \triangleq m.type = RICSubscriptionDeleteRequestType$

$IsRICSubscriptionDeleteResponse(m) \triangleq m.type = RICSubscriptionDeleteResponseType$

$IsRICSubscriptionDeleteFailure(m) \triangleq m.type = RICSubscriptionDeleteFailureType$

$IsRICControlRequest(m) \triangleq m.type = RICControlRequestType$

$IsRICControlResponse(m) \triangleq m.type = RICControlResponseType$

$IsRICControlFailure(m) \triangleq m.type = RICControlFailureType$

$IsRICServiceUpdate(m) \triangleq m.type = RICServiceUpdateType$

$IsE2ConnectionUpdate(m) \triangleq m.type = E2ConnectionUpdateType$

$IsE2ConnectionUpdateAcknowledge(m) \triangleq m.type = E2ConnectionUpdateAcknowledgeType$

$IsE2ConnectionUpdateFailure(m) \triangleq m.type = E2ConnectionUpdateFailureType$

$IsE2NodeConfigurationUpdate(m) \triangleq m.type = E2NodeConfigurationUpdateType$

$IsE2NodeConfigurationUpdateAcknowledge(m) \triangleq m.type = E2NodeConfigurationUpdateAcknowledgeType$

$IsE2NodeConfigurationUpdateFailure(m) \triangleq m.type = E2NodeConfigurationUpdateFailureType$

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 $ValidE2SetupRequest(m) \triangleq$ TRUE

LOCAL $ValidE2SetupResponse(m) \triangleq$ TRUE

LOCAL $ValidE2SetupFailure(m) \triangleq$ TRUE

LOCAL $ValidResetRequest(m) \triangleq$ TRUE

LOCAL $ValidResetResponse(m) \triangleq$ TRUE

LOCAL $ValidRICSubscriptionRequest(m) \triangleq$ TRUE

LOCAL $ValidRICSubscriptionResponse(m) \triangleq$ TRUE

LOCAL $ValidRICSubscriptionFailure(m) \triangleq$ TRUE

LOCAL $ValidRICSubscriptionDeleteRequest(m) \triangleq$ TRUE

LOCAL $ValidRICSubscriptionDeleteResponse(m) \triangleq$ TRUE

LOCAL $ValidRICSubscriptionDeleteFailure(m) \triangleq$ TRUE

LOCAL $ValidRICControlRequest(m) \triangleq$ TRUE

LOCAL $ValidRICControlResponse(m) \triangleq$ TRUE

LOCAL $ValidRICControlFailure(m) \triangleq$ TRUE

LOCAL $ValidRICServiceUpdate(m) \triangleq$ TRUE

LOCAL $ValidE2ConnectionUpdate(m) \triangleq$ TRUE

LOCAL $ValidE2ConnectionUpdateAcknowledge(m) \triangleq$ TRUE

LOCAL $ValidE2ConnectionUpdateFailure(m) \triangleq$ TRUE

LOCAL $ValidE2NodeConfigurationUpdate(m) \triangleq$ TRUE

LOCAL $ValidE2NodeConfigurationUpdateAcknowledge(m) \triangleq$ TRUE

LOCAL $ValidE2NodeConfigurationUpdateFailure(m) \triangleq$ TRUE

This section defines operators for constructing $E2AP$ messages.

LOCAL $SetType(m,\ t) \triangleq [m \text{ EXCEPT } !.type = t]$

$E2SetupRequest(m) \triangleq$
IF $Assert(ValidE2SetupRequest(m),$ "Invalid E2SetupRequest")
THEN $SetType(m,\ E2SetupRequestType)$
ELSE Nil

$E2SetupResponse(m) \triangleq$
IF $Assert(ValidE2SetupResponse(m),$ "Invalid E2SetupResponse")
THEN $SetType(m,\ E2SetupResponseType)$
ELSE Nil

$E2SetupFailure(m) \triangleq$
IF $Assert(ValidE2SetupFailure(m),$ "Invalid E2SetupFailure")
THEN $SetType(m,\ E2SetupFailureType)$
ELSE Nil

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

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

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

$RICSubscriptionResponse(m) \triangleq$
IF $Assert(ValidRICSubscriptionResponse(m),$ "Invalid RICSubscriptionResponse")
THEN $SetType(m,\ RICSubscriptionResponseType)$
ELSE Nil

$RICSubscriptionFailure(m) \triangleq$
IF $Assert(ValidRICSubscriptionFailure(m),$ "Invalid RICSubscriptionFailure")
THEN $SetType(m,\ RICSubscriptionFailureType)$
ELSE Nil

$RICSubscriptionDeleteRequest(m) \triangleq$
IF $Assert(ValidRICSubscriptionDeleteRequest(m),$ "Invalid RICSubscriptionDeleteRequest")
THEN $SetType(m,\ RICSubscriptionDeleteRequestType)$
ELSE Nil

$RICSubscriptionDeleteResponse(m) \triangleq$
IF $Assert(ValidRICSubscriptionDeleteResponse(m),$ "Invalid RICSubscriptionDeleteResponse")
THEN $SetType(m,\ RICSubscriptionDeleteResponseType)$

ELSE Nil

$RICSubscriptionDeleteFailure(m) \triangleq$
IF $Assert(ValidRICSubscriptionDeleteFailure(m),$ "Invalid RICSubscriptionDeleteFailure")
THEN $SetType(m, RICSubscriptionDeleteFailureType)$
ELSE Nil

$RICControlRequest(m) \triangleq$
IF $Assert(ValidRICControlRequest(m),$ "Invalid RICControlRequest")
THEN $SetType(m, RICControlRequestType)$
ELSE Nil

$RICControlResponse(m) \triangleq$
IF $Assert(ValidRICControlResponse(m),$ "Invalid RICControlResponse")
THEN $SetType(m, RICControlResponseType)$
ELSE Nil

$RICControlFailure(m) \triangleq$
IF $Assert(ValidRICControlFailure(m),$ "Invalid RICControlFailure")
THEN $SetType(m, RICControlFailureType)$
ELSE Nil

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

$E2ConnectionUpdate(m) \triangleq$
IF $Assert(ValidE2ConnectionUpdate(m),$ "Invalid E2ConnectionUpdate")
THEN $SetType(m, E2ConnectionUpdateType)$
ELSE Nil

$E2ConnectionUpdateAcknowledge(m) \triangleq$
IF $Assert(ValidE2ConnectionUpdateAcknowledge(m),$ "Invalid E2ConnectionUpdateAcknowledge")
THEN $SetType(m, E2ConnectionUpdateAcknowledgeType)$
ELSE Nil

$E2ConnectionUpdateFailure(m) \triangleq$
IF $Assert(ValidE2ConnectionUpdateFailure(m),$ "Invalid E2ConnectionUpdateFailure")
THEN $SetType(m, E2ConnectionUpdateFailureType)$
ELSE Nil

$E2NodeConfigurationUpdate(m) \triangleq$
IF $Assert(ValidE2NodeConfigurationUpdate(m),$ "Invalid E2NodeConfigurationUpdate")
THEN $SetType(m, E2NodeConfigurationUpdateType)$
ELSE Nil

$E2NodeConfigurationUpdateAcknowledge(m) \triangleq$

IF $Assert(ValidE2NodeConfigurationUpdateAcknowledge(m)$, "Invalid E2NodeConfigurationUpdateAcknow
THEN $SetType(m, E2NodeConfigurationUpdateAcknowledgeType)$
ELSE Nil

$E2NodeConfigurationUpdateFailure(m) \triangleq$
IF $Assert(ValidE2NodeConfigurationUpdateFailure(m)$, "Invalid E2NodeConfigurationUpdateFailure")
THEN $SetType(m, E2NodeConfigurationUpdateFailureType)$
ELSE Nil

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

LOCAL $Messages \triangleq$ INSTANCE $Messages$

MODULE *Client*

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

MODULE *Send*

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

$E2SetupRequest(c, m) \triangleq$
$\land SCTP!Client!Send(c, Messages!E2SetupResponse(m))$

$ResetRequest(c, m) \triangleq$
$\land SCTP!Client!Send(c, Messages!ResetRequest(m))$

$ResetResponse(c, m) \triangleq$
$\land SCTP!Client!Reply(c, Messages!ResetResponse(m))$

Instantiate the $E2AP!Client!Send$ module

$Send \triangleq$ INSTANCE $Send$

MODULE *Receive*

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

$E2SetupResponse(c, h(\_, \_)) \triangleq$
$SCTP!Server!Handle(c,$ LAMBDA $x, m :$
$\land Messages!IsE2SetupResponse(m)$
$\land SCTP!Client!Receive(c)$
$\land h(c, m))$

$ResetRequest(c, h(\_, \_)) \triangleq$
$SCTP!Server!Handle(c,$ LAMBDA $x, m :$

$\land$ *Messages*!*IsResetRequest*(m)
$\land$ *SCTP*!*Client*!*Receive*(c)
$\land$ $h(c,\ m)$)

ResetResponse(c, $h(\_, \_)$) $\triangleq$
SCTP!*Server*!*Handle*(c, LAMBDA x, m :
$\land$ *Messages*!*IsResetResponse*(m)
$\land$ *SCTP*!*Client*!*Receive*(c)
$\land$ $h(c,\ m)$)

Instantiate the *E2AP*!*Client*!*Receive* module
Receive $\triangleq$ INSTANCE *Receive*

Connect(s, d) $\triangleq$ *SCTP*!*Client*!*Connect*(s, d)

Disconnect(c) $\triangleq$ *SCTP*!*Client*!*Disconnect*(c)

Provides operators for the *E2AP* client
Client $\triangleq$ 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.

MODULE *Send*

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

E2SetupResponse(c, m) $\triangleq$
$\land$ *SCTP*!*Server*!*Reply*(c, *Messages*!*E2SetupResponse*(m))

ResetRequest(c, m) $\triangleq$
$\land$ *SCTP*!*Server*!*Send*(c, *Messages*!*ResetRequest*(m))

ResetResponse(c, m) $\triangleq$
$\land$ *SCTP*!*Server*!*Reply*(c, *Messages*!*ResetResponse*(m))

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

MODULE *Receive*

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

$E2SetupRequest(c,\ h(\_,\ \_)) \triangleq$
$SCTP!Server!Handle(c,$ LAMBDA $x,\ m :$
$\land\ Messages!IsE2SetupRequest(m)$
$\land\ SCTP!Server!Receive(c)$
$\land\ h(c,\ m))$

$ResetRequest(c,\ h(\_,\ \_)) \triangleq$
$SCTP!Server!Handle(c,$ LAMBDA $x,\ m :$
$\land\ Messages!IsResetRequest(m)$
$\land\ SCTP!Server!Receive(c)$
$\land\ h(c,\ m))$

$ResetResponse(c,\ h(\_,\ \_)) \triangleq$
$SCTP!Server!Handle(c,$ LAMBDA $x,\ m :$
$\land\ Messages!IsResetResponse(m)$
$\land\ SCTP!Server!Receive(c)$
$\land\ h(c,\ m))$

Instantiate the $E2AP!Server!Receive$ module
$Receive \triangleq$ INSTANCE $Receive$

Starts a new $E2AP$ server
$Serve(s) \triangleq SCTP!Server!Start(s)$

Stops the given $E2AP$ server
$Stop(s) \triangleq SCTP!Server!Stop(s)$

Provides operators for the $E2AP$ server
$Server \triangleq$ INSTANCE $Server$

The set of all running $E2AP$ servers
$Servers \triangleq SCTP!Servers$

The set of all open $E2AP$ connections
$Connections \triangleq SCTP!Connections$

$Init \triangleq SCTP!Init$

$Next \triangleq SCTP!Next$

---

* Modification History
* Last modified *Mon Sep* 13 12:35:51 *PDT* 2021 by *jordanhalterman*
* *Created Mon Sep* 13 10:53:17 *PDT* 2021 by *jordanhalterman*