The Complete Formal Description of the SNMPv3 Entity Using Action Semantics

Elias P. Duarte Jr.
Martin A. Musicante

Federal University of Paraná
Dept. Informatics
Curitiba PR Brazil
`{elias,mam}@inf.ufpr.br`

Diógenes Cogo Furlan

Tuiuti University of Paraná
FaCET
Curitiba PR Brazil
`diogenes.furlan@utp.br`

`Technical Report # RT001/2002, Federal University of Parana, Dept. Informatics, http://www.inf.ufpr.br/info/techrep/index.html`

1 Actions

1.1 Entity

- Manager-daemon :: action

```
(1)  Manager-daemon =
        | | subordinate a dispatcher then bind “dispatcher” to it
        | and
        | | subordinate an application then bind “CR” to it
        | and initiate user-applications
        | and initiate MPS
        | and initiate SS
        hence
        | | send a message[to the agent bound to “dispatcher”]
        | |                  [containing closure abstraction of Dispatcher-daemon]
        | and
        | | send a message[to the agent bound to “CR”][containing closure abstraction of CR-daemon]
        | and activate user-applications
        | and activate MPS
        | and activate SS
```

- Agent-daemon :: action

```
(2)  Agent-daemon =
         | | subordinate a dispatcher then bind “dispatcher” to it
         | and
         | | subordinate an application then bind “NR” to it
         | and initiate user-applications
         | and initiate MPS
         | and initiate SS
         | and initiate ACS
         hence
         | | send a message[to the agent bound to “dispatcher”]
         | |               [containing closure abstraction of Dispatcher-daemon]
         | and
         | | send a message[to the agent bound to “NR”][containing closure abstraction of NR-daemon]
         | and activate user-applications
         | and activate MPS
         | and activate SS
         | and activate ACS
```

- Dispatcher-daemon :: action

```
(3)  Dispatcher-daemon =
         | initialize LCD-Dispatcher
         hence unfolding
         | | | procedure Dispatcher
         | | trap complete
         | and then unfold
```

- user-application _ :: integer $\to$ action

```
(4)  user-application X:integer =
         | initialize LCD-Generate
         hence
         | ...
         | procedure CommandGenerator
         | ...

(5)  user-application X:integer =
         | initialize LCD-Generate
         hence
         | ...
         | procedure NotificationOriginator
         | ...
```

- CommandResponder-daemon :: action

```
(6)  CommandResponder-daemon =
        | | initialize MIBs
        | and
        | | initialize LCD-Process
        | and
        | | register CR
        hence unfolding
        | | | procedure CommandResponder
        | | trap complete
        | and then unfold
```

- NotificationReceiver-daemon :: action

```
(7)  NotificationReceiver-daemon =
        | | initialize LCD-Process
        | and
        | | register NR
        hence unfolding
        | | | procedure NotificationReceiver
        | | trap complete
        | and then unfold
```

- initialize LCD-Dispatcher :: action

```
(8)  initialize LCD-Dispatcher =
        allocate a cell then bind "sender" to it
```

- initialize LCD-Generate :: action

```
(9)  initialize LCD-Generate =
        | rebind
        and allocate a cell then bind "transpDom" to it
        and allocate a cell then bind "transpAdd" to it
        and allocate a cell then bind "MPModel" to it
        and allocate a cell then bind "secModel" to it
        and allocate a cell then bind "secName" to it
        and allocate a cell then bind "secLevel" to it
        and allocate a cell then bind "pduVersion" to it
        and allocate a cell then bind "contextEngID" to it
        and allocate a cell then bind "contextName" to it
```

- initialize LCD-Process :: action

```
(10) initialize LCD-Process =
        | rebind
        and allocate a cell then bind “MPModel” to it
        and allocate a cell then bind “secModel” to it
        and allocate a cell then bind “secName” to it
        and allocate a cell then bind “secLevel” to it
        and allocate a cell then bind “pduVersion” to it
        and allocate a cell then bind “contextEngID” to it
        and allocate a cell then bind “contextName” to it
        and allocate a cell then bind “request-id” to it
        and allocate a cell then bind “non-repeaters” to it
        and allocate a cell then bind “max-repetitions” to it
        and allocate a cell then bind “variable-bindings” to it
```

- initialize MIBs :: action

```
(11) initialize MIBs = □
```

- register CR :: action

```
(12) register CR =
        | get(“snmpEngineID”)
        then
        | | associate performing-agent with (it, Get)
        | and
        | | associate performing-agent with (it, GetNext)
        | and
        | | associate performing-agent with (it, GetBulk)
        | and
        | | associate performing-agent with (it, Set)
```

- register NR :: action

```
(13) register NR =
        | get(“snmpEngineID”)
        then
        | | associate performing-agent with (it, Inform)
        | and
        | | associate performing-agent with (it, Trap)
```

- associate _ with _ :: agent, tuple $\rightarrow$ action

(14) associate A:agent with (C:contextEngineId, O:pduType) = □

- initiate user-applications :: action

(15) initiate user-applications = □

- initiate MPS :: action

(16) initiate MPS = □

- initiate SS :: action

(17) initiate SS = □

- initiate ACS :: action

(18) initiate ACS = □

- activate user-applications :: action

(19) activate user-applications = □

- activate MPS :: action

(20) activate MPS = □

- activate SS :: action

(21) activate SS = □

- activate ACS :: action

(22) activate ACS = □

1.2 Dispatcher

- procedure Dispatcher :: action

(23) procedure Dispatcher =

```
    | accept a message[from an agent][containing a SDU]
    then
    | | procedure Disp send-request
    | or
    | | procedure Disp send-response
    | or
    | | procedure Disp receive
```

- procedure Disp send-request :: action[receiving tuple]

```
(24) procedure Disp send-request =
        | | give (the given tuple)[sendPdu-SDU]
        |  then
        |  | procedure Disp send-request A
        |  then
        | | send a message[to the MPModel associated with the given messageProcessingModel#3]
        | |                [containing (the given tuple)[prepareOutgoingMsgIn-SDU]]
       and then
        | | accept contents of a message[from an MPModel][containing a tuple[prepareOutgoingMsgOut-S
        |  then
        |  | procedure Disp send-request B
        |  then
        | | send a message[to the Dispatcher associated with (the given transportDomain#1,
        | |                the given transportAddress#2)][containing the given Network-MSG#3]
```

- procedure Disp send-request A :: action[receiving sendPdu-SDU][giving prepareOutgoingMsgIn-SDU]

```
(25) procedure Disp send-request A =
        | | check (the MPModel associated with the given messageProcessingModel#3 is an agent)
        |  or
        | | | check not (the MPModel associated with the given messageProcessingModel#3 is an agent)
        | |  and then
        | |  | send noMPModel back to sender
        | |  and then
        | | | escape
       and then
        | regive and generate SendPduHandle
```

- procedure Disp send-request B :: action[receiving prepareOutgoingMsgOut-SDU]

```
(26) procedure Disp send-request B =
        | | | check (the given statusInformation#1 is success)
        | |  and then
        | | | send the generated sendPduHandle back to sender
        |  or
        | | | check not (the given statusInformation#1 is success)
        | |  and then
        | |  | send the given statusInformation#1 back to sender
        | |  and then
        | | | escape
       and then
        | give (the given transportDomain#2, the given transportAddress#3, the given Network-MSG#4
```

• procedure Disp send-response :: action[receiving tuple]

```
(27) procedure Disp send-response =
        | | give (the given tuple)[returnResponsePdu-SDU]
        | then
        | | send a message[to the MPModel associated with the given messageProcessingModel#1]
        | |                 [containing (the given tuple)[prepareResponseMsgIn-SDU]]
       and then
        | | accept contents of a message[from an MPModel][containing a tuple[prepareResponseMsgOut-
        | then
        | | procedure Disp send-response B
        | then
        | | send a message[to the Dispatcher associated with (the given transportDomain#1,
        | |                  the given transportAddress#2)][containing the given Network-MSG#3]
```

• procedure Disp send-response B :: action[receiving prepareResponseMsgOut-SDU]

```
(28) procedure Disp send-response B =

     | | check (the given Result#1 is success)
     | or
     | | | check not (the given Result#1 is success)
     | | and then
     | | | send the given Result#1 back to sender
     | | and then
     | | | escape
     and then
     | give (the given transportDomain#2, the given transportAddress#3, the given Network-MSG#4)
```

• procedure Disp receive :: action[receiving tuple]

```
(29) procedure Disp receive =
| | give (the given tuple)[Network-MSG]
|  then
|  | procedure Disp receive A
|  then
|  | send a message[to the MPModel associated with the given messageProcessingModel#1]
|  |                    [containing (the rest of the given tuple)[prepareDataElementsIn-SDU]]
and then
| | accept contents of a message[from an MPModel][containing a tuple[prepareDataElementsOut-
|  then
|  | | | procedure Disp receive-request B
|  | |  then
|  | | | send a message[to the given Application#1]
|  | | |                    [containing (the rest of the given tuple)[processPdu-SDU]]
|  |  or
|  | | | procedure Disp receive-response B
|  | |  then
|  | | | send a message[to the given Application#1]
| | | |                    [containing (the rest of the given tuple)[processResponsePdu-SDU]]
```

- procedure Disp receive A :: action[receiving Network-MSG]

(30) procedure Disp receive A =

```
    | increment("snmpInPkts")
    and then
    | | | give the messageProcessingModel extracted from the given Network-MSG
    | |  trap
    | | | | increment("snmpInASNParseErrs")
    | | |  and then
    | | | | escape
    |  then
    | | | give (the MPModel associated with the given messageProcessingModel)
    | |  or
    | | | | check not (the MPModel associated with the given messageProcessingModel is an agent)
    | | |  and then
    | | | | increment("snmpInBadVersions")
    | | |  and then
    | | | | escape
    and then
    | give (the transportDomain extracted from the given Network-MSG,
    |       the transportAddress extracted from the given Network-MSG)
    and then
    | regive
```

- procedure Disp receive-request B :: action[receiving prepareDataElementsOut-SDU]

(31) procedure Disp receive-request B =

```
| | | check not (the given Result#1 is sucess)
| | and then escape
| or
| | | check (the given Result#1 is sucess)
| | and then
| | | check (the given sendPduHandle#11 is none)
| | and then
| | | | give the Application associated with (the given contextEngineID#7, the given pduType#10
| | | trap
| | | | | increment("snmpUnknownPDUHandlers")
| | | | and then
| | | | | | regive and get("snmpUnknownPDUHandlers")
| | | | | then
| | | | | | give (the given messageProcessingModel#2, securityModel#3, securityName#4,
| | | | | |         securityLevel#5, pduVersion#6, contextEngineID#7, contextName#8, PDU#9,
| | | | | |         maxSizeResponseScopedPdu#12, stateReference#14,
| | | | | |         par of(noApplication, the given ObjectValue#15))
| | | | | then
| | | | | | procedure Disp send-response
| | | | and then
| | | | | escape
and then
| give (the given messageProcessingModel#2, securityModel#3, securityName#4, securityLevel#
|       pduVersion#6, contextEngineID#7, contextName#8, PDU#9,
|       maxSizeResponseScopedPdu#12, stateReference#14)
```

- procedure Disp receive-response B :: action[receiving prepareDataElementsOut-SDU]

```
(32) procedure Disp receive-response B =
          | | | check not (the given Result#1 is sucess)
          | | and then
          | | | escape
          | or
          | | | check (the given Result#1 is sucess)
          | | and then
          | | | check not (the given sendPduHandle#11 is none)
          | | and then
          | | | | give the Application associated with the given sendPduHandle#11
          | | | trap
          | | | | | increment("snmpUnknownPDUHandlers")
          | | | | and then
          | | | | | escape
          and then
          | give (the given messageProcessingModel#2, securityModel#3, securityName#4, securityLevel#
          |      pduVersion#6, contextEngineID#7, contextName#8, PDU#9, statusInformation#13,
          |      sendPduHandle#11)
```

- generate sendPduHandle :: action[giving sendPduHandle]

```
(33) generate sendPduHandle = □
```

1.3 Standard Applications

- procedure CommandGenerator :: action

```
(34) procedure CommandGenerator =
          | | procedure Aplic send-request
          | then
          | | send a message[to Dispatcher of entity][containing (the given tuple)[sendPdu-SDU]]
          | then
          | | accept contents of a message[from Dispatcher of entity][containing a datum]
          | then
          | | | ifnot (it is a sendPduHandle) generate error it
          | | and then
          | | | cache it as "sendPduHandle"
          and then
          | | accept contents of a message[from Dispatcher of entity]
          | |                                  [containing a tuple[processResponsePdu-SDU]]
          | then
          | | procedure Aplic receive-response
```

- procedure CommandResponder :: action

(35) procedure CommandResponder =

```
    | accept contents of a message[from Dispatcher of entity][containing a tuple[processPdu-SDU]]
    then
    | procedure Aplic receive-request
    then
    | | give the cached “variable-bindings” then map using abstraction of isAccessAllowed
    | and then
    | | operationResponse the given opTag#5
    then
    | procedure Aplic send-response
    then
    | send a message[to Dispatcher of entity][containing (the given tuple)[returnResponsePdu-SDU]]
```

- procedure NotificationOriginator :: action

(36) procedure NotificationOriginator =

```
    | | give the given list#2
    | then
    | | map using abstraction of isAccessAllowed
    and then
    | | procedure Aplic send-request
    | then
    | | send a message[to Dispatcher of entity][containing (the given tuple)[sendPdu-SDU]]
    and then
    | | | check (the given opTag#1 is Inform)
    | | and then
    | | | | accept contents of a message[from Dispatcher of entity]
    | | | |                                   [containing a tuple[processResponsePdu-SDU]]
    | | | then
    | | | | give (the error-status of the given Pdu#8, the error-index of the given Pdu#8,
    | | | |        the given variable-bindings of the given Pdu#8)
    | or
    | | check (the given opTag#1 is Trap)
```

- procedure NotificationReceiver :: action

(37) procedure NotificationReceiver =

```
    | accept contents of a message[from Dispatcher of entity][containing a tuple[processPdu-SDU]]
    then
    | procedure Aplic receive-request
    then
    | | | check (the given opTag#5 is Inform)
    | | and then
    | | | | operationResponse Inform
    | | | then
    | | | | procedure Aplic send-response
    | | | then
    | | | | send a message[to Dispatcher of entity][containing (the given tuple)[returnResponsePdu-SD
    | or
    | | | check (the given opTag#5 is Trap)
    | | and then
    | | | operationResponse Trap
```

• procedure Aplic send-request :: action[giving sendPdu-SDU]

(38) procedure Aplic send-request =

```
    | give (the cached “transpDom”, the cached “transpAdd”, the cached “MPModel”,
    |       the cached “secModel”, the cached “secName”, the cached “secLevel”,
    |       the cached “pduVersion”, the cached “contEngID”, the cached “contName”)
    and
    | | | check not (the given opTag#1 is GetBulk)
    | | and then
    | | | operationRequest the given opTag#1 with (0, 0, the given list#2)
    | or
    | | | check (the given opTag#1 is GetBulk)
    | | and then
    | | | operationRequest GetBulk with (the given non-repeaters#3, the given max-repetitions#4,
    | | |                                the given list#2)
    and
    | give not (the given opTag#1 is Trap)
```

• procedure Aplic receive-response :: action[receiving processResponsePdu-SDU]

(39) procedure Aplic receive-response =

```
  | | ifnot (the given messageProcessingModel#1 is the cached "MPModel")
  | |       generate error discardByMPModel
  | and
  | | ifnot (the given securityModel#2 is the cached "secModel")
  | |       generate error discardBySecModel
  | and
  | | ifnot (the given securityName#3 is the cached "secName")
  | |       generate error discardBySecName
  | and
  | | ifnot (the given pduVersion#5 is the cached "pduVersion")
  | |       generate error discardByPduVersion
  | and
  | | ifnot (the given contextEngineID#6 is the cached "contEngID")
  | |       generate error discardByContextEngID
  | and
  | | ifnot (the given contextName#7 is the cached "contName")
  | |       generate error discardByContextName
  | and
  | | ifnot (the request-id of the given Pdu#8 is the generated "request-id")
  | |       generate error discardByRequestID
  and then
  | give (error-status of the given Pdu#8, error-index of the given Pdu#8,
  |       variable-bindings of the given Pdu#8)
```

- procedure Aplic receive-request :: action[receiving processPdu-SDU][giving (request-id,error-status,Index,list,opTag)]

(40) procedure Aplic receive-request =

```
    ifnot (the performing-agent is associated with the operation of the given PDU#8)
        generate error discardByOperation
and then
    give (request-id of the given PDU#8, error-status of the given PDU#8,
          error-index of the given PDU#8, variable-bindings of the given PDU#8,
          operation of the given PDU#8)
  then
      regive and cache the splitted PDU
and then
    cache the given messageProcessingModel as "MPModel"
  and
    cache the given securityModel as "secModel"
  and
    cache the given securityName as "secName"
  and
    cache the given securityLevel as "secLevel"
  and
    cache the given pduVersion as "pduVersion"
  and
    cache the given contextEngineID as "contextEngID"
  and
    cache the given contextName as "contextName"
```

- procedure Aplic send-response :: action[giving returnResponsePdu-SDU]

(41) procedure Aplic send-response =

```
    give (the cached "MPModel",
          the cached "secModel", the cached "secName", the cached "secLevel",
          the cached "pduVersion", the cached "contEngID", the cached "contName")
and then
    give Pdu of (the cached "request-id", the given errorStatus#1,
                 the given Index#2, the given VarBindList#3) with tag Response
and then
    give (size of scopedPdu, the cached "stateReference", the given errorStatus#1)
```

1.4 Protocol Operations

- operationRequest _ with (_, _, _) :: opTag, Index, Index, tuple → action

(42) operationRequest Op:opTag with (S:Index, I:Index, T:tuple) =

```
    | | generate request-id
    | and
    | | generate VarBindList from $T$
    then
    | give the PDU of (the given requestId#1, $S$, $I$, the given VarBindList#2) with tag $Op$
```

- generate VarBindList from _ :: tuple → action

```
(43) generate VarBindList from $N$:list of ObjectName$^+$ =
    | give $N$
    then
    | map using abstraction of emptyVarBind

(44) generate VarBindList from $V$:VarBindList = regive

(45) generate VarBindList from $M$:Notification-Macro =
    | | | get("sysUpTime.0")
    | | then
    | | | give list of ("sysUpTime.0", the given ObjectValue)
    | and
    | | | get("snmpTrapOID.0")
    | | then
    | | | give list of ("snmpTrapOID.0", the given ObjectValue)
    | and
    | | | generate ObjectNameList from $M$
    | | then
    | | | map using abstraction of consultVarBind
    then
    | give concatenation(the given list#1, the given list#2, the given list#3)
```

- operationResponse _ :: opTag → action

```
(46) operationResponse Get =
    | | setError(noError, 0)
    | and
    | | | give the cached "variable-bindings"
    | | then
    | | | map using abstraction of consultVarBind
    trap
    | | setError(genErr, the given Index)
    | and
    | | give the cached "variable-bindings"
```

```
(47) operationResponse GetNext =
        | | setError(noError, 0)
        | and
        | | | give the cached “variable-bindings”
        | | then
        | | | map using abstraction of consultNextVarBind
        trap
        | | setError(genErr, the given Index)
        | and
        | | give the cached “variable-bindings”

(48) operationResponse GetBulk =
        | | setError(noError, 0)
        | and
        | | | | give min(the cached “non-repeaters”, count items of the cached “variable-bindings”)
        | | | and
        | | | | give the cached “variable-bindings”
        | | then
        | | | break(the given list#2, the given integer#1)
        | | then
        | | | | | give the given list#1
        | | | | then
        | | | | | map using abstraction of consultNextVarBind
        | | | and then
        | | | | | | give the cached “max-repetitions”
        | | | | | and
        | | | | | | give the given list#2
        | | | | then
        | | | | | map with repetition using abstraction of consultNextVarBind
        | | then
        | | | give concatenation(the given list#1, the given list#2)
        trap
        | | setError(genErr, the given Index)
        | and
        | | give the cached “variable-bindings”

(49) operationResponse Set =
      |  |  | give the cached “variable-bindings”
      |  | then
      |  |  | map using abstraction of validateVarBind
      |  | then
      |  |  |  |  | setError(noError, 0)
      |  |  |  | and
      |  |  |  |  | map using abstraction of updateVarBind
      |  |  | trap
      |  |  |  |  | setError(commitFailed, the given Index)
      |  |  |  | and
      |  |  |  |  |  | break(the cached “variable-bindings”, the given Index)
      |  |  |  |  | then
      |  |  |  |  |  | map using abstraction of undoVarBind
      |  |  | trap
      |  |  |  | setError(undoFailed, 0)
      | trap
      |  | setError(the given errorStatus#1, the given Index#2)
    then
      | give (the given errorStatus#1, the given Index#2, the cached “variable-bindings”)
```

```
(50) operationResponse Inform =
       | setError(noError, 0)
     and
       | give the cached “variable-bindings”
     and
       |  | give the Application associated with NotificationTypeId of the cached “variable-bindings”
       | then
       |  | send a message [to the given Application][containing the cached “variable-bindings”]
```

```
(51) operationResponse Trap =
       | give the Application associated with NotificationTypeId of the cached “variable-bindings”
     then
       | send a message [to the given Application][containing the cached “variable-bindings”]
```

- generate request-id :: action[giving requestId]

(52) generate request-id = $\Box$

- generate ObjectNameList from _ :: Notification-Macro $\rightarrow$ action[giving ObjectNameList]

(53) generate ObjectNameList from M:Notification-Macro = $\Box$

1.5 *VarBind* Operations

- emptyVarBind :: action[receiving VarBind][giving VarBind]

```
(54) emptyVarBind =
        give (the given ObjectName#1, unSpecified)
```

- consultVarBind :: action[receiving VarBind][giving VarBind]

```
(55) consultVarBind =
        | | check (existsObject the given ObjectName#1)
        | and then
        | | | | check not (existsInstance the given ObjectName#1)
        | | | and then
        | | | | | give the given ObjectName#1
        | | | | and
        | | | | | get(the given ObjectName#1)
        | | else
        | | | give (the given ObjectName#1, noSuchInstance)
        else
        | give (the given ObjectName#1, noSuchObject)
```

- consultNextVarBind :: action[receiving VarBind][giving VarBind]

```
(56) consultNextVarBind =
        | | give the next to the given ObjectName#1
        | then
        | | | regive
        | | and
        | | | get(the given ObjectName)
        else
        | give (the given ObjectName#1, endOfMibView)
```

- validateVarBind :: action[receiving VarBind]

```
(57) validateVarBind =
  | ifnot (existsObject the given ObjectName#1)
  |       generate error notWritable
  and then
  | ifnot (type(the given VarBind) is type(the given ObjectName#1))
  |       generate error wrongType
  and then
  | ifnot (length(the given VarBind) is length(the given ObjectName#1))
  |       generate error wrongLength
  and then
  | ifnot (encoding(the given VarBind) is encoding(the given ObjectName#1))
  |       generate error wrongEncoding
  and then
  | ifnot (alwaysAtrib(the given ObjectName#1, the given ObjectSyntax#2))
  |       generate error wrongValue
  and then
  | ifnot (alwaysCreate(the given ObjectName#1))
  |       generate error noCreation
  and then
  | ifnot (nowCreate(the given ObjectName#1))
  |       generate error inconsistentName
  and then
  | ifnot (not access(the given ObjectName#1) is READ-ONLY and
  |       not access(the given ObjectName#1) is NOT-ACCESSIBLE)
  |       generate error notWritable
  and then
  | ifnot (nowAtrib(the given ObjectName#1, the given ObjectSyntax#2))
  |       generate error inconsistentValue
```

- updateVarBind :: action[receiving VarBind][giving VarBind]

```
(58) updateVarBind =
       set(the given ObjectName#1, the given ObjectSyntax#2)
```

- isAccessAllowed :: action

```
(59) isAccessAllowed =
| give (the cached “secModel”, the cached “secName”, the cached “secLevel”,
|       class of the cached “pduType”, the cached “contextName”, the given ObjectName#1)
then
| send a message[to ACModel of entity][containing then]
then
| accept contents of message[from ACModel of entity][containing an StatusInformation]
then
| | | check all (it is noSuchView, it is noAccessEntry, it is noGroupNome)
| | and then
| | | | setError(authorizationError, 0) and give the cached “variable-bidings”
| | | then escape with them
| else
| | | check (it is noSuchContext)
| | and then
| | | | setError(0,0) and give empty-list
| | | then escape with them
| else
| | | setError(genError,0) and give the cached “variable-bidings”
| | then escape with them
```

1.6 Management Information Operations

- get(_) :: ObjectName $\to$ action[giving ObjectValue]

(60) get(N:ObjectName) = $\Box$

- set(_,_) :: ObjectName, ObjectSyntax $\to$ action

(61) set(N:ObjectName,S:ObjectSyntax) = $\Box$

1.7 Auxiliar

- map using _ :: abstraction $\to$ action[receiving list]

(62) map using A:abstraction =

```
unfolding
      | | | check ( the given list is empty-list )
      | | and then
      | | | give it
      | or
      | | | check not ( the given list is empty-list )
      | | and then
      | | | | | | give head of the given list
      | | | | | then
      | | | | | | enact $A$
      | | | | | then
      | | | | | | give list of the given tuple
      | | | | and
      | | | | | | give tail of the given list
      | | | | | then unfold
      | | | then
      | | | | give concatenation(the given list#1, the given list#2)
```

• map with repetition using _ :: abstraction → action[receiving (natural, list)]

(63) map with repetition using A:abstraction =

```
unfolding
      | | | check ( the given natural#1 is 0 )
      | | and then
      | | | give the empty-list
      | or
      | | | check not ( the given natural#1 is greater than 0 )
      | | and then
      | | | | | give difference(the given natural#1, 1)
      | | | | and
      | | | | | | give the given list#2
      | | | | | then
      | | | | | | map using $A$
      | | | then
      | | | | | give the given list#2
      | | | | and
      | | | | | unfold
      | | | then
      | | | | give concatenation(the given list#1, the given list#2)
```

• break _ :: (list, integer) → action[giving (list, list)]

(64) break (L:list, I:integer) =

```
        | give 0 and give $L$
        then
        | unfolding
        | | | | ckeck (the given list#2 is empty-list)
        | | | and then
        | | | | give (empty-list, empty-list)
        | | or
        | | | | | ckeck (the given integer#1 is $I$)
        | | | | and then
        | | | | | give (empty-list, the given list#2)
        | | | or
        | | | | | ckeck (the given integer#1 is less than $I$)
        | | | | and then
        | | | | | | | give list of head of the given list#2
        | | | | | | and
        | | | | | | | | | give sum(the given integer#1, 1)
        | | | | | | | | and
        | | | | | | | | | give tail of the given list#2
        | | | | | | | then
        | | | | | | | | unfold
        | | | | | then
        | | | | | | give (concatenation(the given list#1, the given list#2), the given list#3)
```

- accept _ :: message $\rightarrow$ action[giving tuple]

(65) accept M:message =

```
        | receive $M$
        then
        | | cache the sender of it as "sender"
        | and
        | | give the contents of it
```

- accept contents of _ :: message $\rightarrow$ action[giving tuple]

(66) accept contents of M:message =

```
        | receive $M$
        then
        | give the contents of it
```

- ifnot _ generate error _ :: yielder[of truth-value], errorStatus $\rightarrow$ action[receiving ObjectName]

(67) ifnot Y:yielder generate error E:errorStatus =

```
  | check $Y$
  or
  | | check not $Y$
  | and then
  | | escape with ($E$, index of the given ObjectName in the cached "variable-bindings")
```

- ifnot _ generate error _ :: yielder[of truth-value], errorIndication → action

```
(68) ifnot $Y$:yielder generate error $E$:errorIndication =
       | check $Y$
       or
       | | check not $Y$
       | and then
       | | escape with $E$
```

- increment(_) :: token → action

```
(69) increment($N$:ObjectName) =
       | get($N$)
       then
       | set($N$, sum(it, 1))

(70) increment($K$:token) =
       store the sum(the integer stored in the cell bound to $K$, 1) in the cell bound to $K$
```

- cache _ as _ :: tuple, token → action

```
(71) cache $D$:tuple as $K$:token =
       store $D$ in the cell bound to $K$
```

- cache the splitted PDU :: action

```
(72) cache the splitted PDU =
       | cache the given requestId#1 as "request-id"
       and
       | cache max(0, the given Index#2) as "non-repeaters"
       and
       | cache the given Index#3 as "max-repetitions"
       and
       | cache the given VarBindList#4 as "variable-bindings"
```

- setError _ :: (errorStatus, Index) → action[giving (errorStatus, Index)]

```
(73) setError(S:errorStatus, I:Index) =
       | give (S, I)
```

- send _ back to sender :: tuple → action

(74) send T:tuple back to sender =
 send a message[to the cached "sender"][containing T]

2 Productors

2.1 Entity

- Dispatcher of entity :: yielder[of agent]

(75) Dispatcher of entity = give the agent bound to "dispatcher"

- ACModel of entity :: yielder[of agent]

(76) ACModel of entity = $\square$

- the generated request-id :: yielder[of requestId]

(77) the generated request-id = $\square$

- the generated sendPduHandle :: yielder[of sendPduHandle]

(78) the generated sendPduHandle = $\square$

- the _ extracted from _ :: tuple, tuple $\rightarrow$ yielder[of tuple]

(79) the $D \leq$ messageProcessingModel extracted from N:Network-MSG = $\square$

(80) the $D \leq$ transportDomain extracted from N:Network-MSG = $\square$

(81) the $D \leq$ transportAddress extracted from N:Network-MSG = $\square$

- the _ associated with _ :: agent, tuple $\rightarrow$ yielder[of agent]

(82) the $A \leq$ Application associated with M:Notification-Macro = $\square$

(83) the $A \leq$ Dispatcher associated with N:transportData = $\square$

(84) the $A \leq$ MPModel associated with M:messageProcessingModel = $\square$

(85) the $A \leq$ Application associated with (C:contextEngineID, O:pduType) = $\square$

(86) the $A \leq$ Application associated with H:sendPduHandle = $\square$

- the _ is associated with _ :: agent, opTag $\rightarrow$ yielder[of truth-value]

(87) the A:agent is associated with O:opTag = $\square$

2.2 Management Information Operations

- existsObject _ :: ObjectName $\rightarrow$ yielder[of truth-value]

(88) existsObject N:ObjectName = $\square$

- existsInstance _ :: ObjectName $\rightarrow$ yielder[of truth-value]

(89) existsInstance N:ObjectName = $\square$

- the next to _ :: ObjectName $\rightarrow$ yielder[of ObjectName]

(90) the next to N:ObjectName = $\square$

- access (_) :: ObjectName $\rightarrow$ yielder

(91) access(N:ObjectName) = $\square$

- type(_) :: tuple $\rightarrow$ yielder

(92) type(V:VarBind) = $\square$

(93) type(N:ObjectName) = $\square$

- length(_) :: tuple $\rightarrow$ yielder[of number]

(94) length(V:VarBind) = $\square$

(95) length(N:ObjectName) = $\square$

- encoding(_) :: tuple $\rightarrow$ yielder

(96) encoding(V:VarBind) = $\square$

(97) encoding(N:ObjectName) = $\square$

- alwaysCreate(_) :: ObjectName $\rightarrow$ yielder[of truth-value]

(98) alwaysCreate(N:ObjectName) = $\square$

- nowCreate(_) :: ObjectName $\rightarrow$ yielder[of truth-value]

(99) nowCreate(N:ObjectName) = $\square$

- alwaysAtrib(_,_) :: ObjectName, ObjectSyntax $\rightarrow$ yielder[of truth-value]

(100) alwaysAtrib(N:ObjectName, S:ObjectSyntax) = $\square$

- nowAtrib(_) :: ObjectName, ObjectSyntax $\rightarrow$ yielder[of truth-value]

(101) nowAtrib(N:ObjectName, S:ObjectSyntax) = $\square$

2.3 Auxiliar

- the cached _ :: token $\rightarrow$ yielder[of datum]

(102) the cached K:token =
the datum stored in the cell bound to K

- class of _ :: opTag $\rightarrow$ yielder

(103) class of O:opTag = $\square$

- index of _ in _ :: ObjectName, VarBindList $\rightarrow$ yielder[of Index]

(104) index of N:ObjectName in V:VarBindList = $\square$

- NotificationTypeId of _ :: VarBindList $\rightarrow$ yielder[of ObjectValue]

(105) NotificationTypeId of V:VarBindList = $\square$

- size of scopedPdu :: yielder[of integer]

(106) size of scopedPdu = $\square$

3 Data

3.1 PDU

(107) opTag = Get | GetNext | GetBulk | Set | Inform | Trap | Response | $\square$ (*individual*)

(108) PDU of (R:requestId, S:errorStatus, I:Index, V:VarBindList) with tag T:tag $\rightarrow$ PDU

(109) request-id of PDU of (R,S,I,V) with tag $T = R$

(110) error-status of PDU of (R,S,I,V) with tag $T = S$

(111) error-index of PDU of (R,S,I,V) with tag $T = I$

(112) variable-bindings of PDU of (R,S,I,V) with tag $T = V$

(113) operation of PDU of (R,S,I,V) with tag $T = T$

- _ tagged with _ :: PDU, tag $\rightarrow$ PDU

(114) P:PDU tagged with Op:opTag =
if operation of P is Op then P else nothing

3.2 SDU

(115) SDU $\geq$ sendPdu-SDU | prepareOutgoingMsgIn-SDU | prepareOutgoingMsgOut-SDU | returnResponsePdu-SDU | prepareResponseMsgIn-SDU | prepareResponseMsgOut-SDU |
prepareDataElementsIn-SDU | prepareDataElementsOut-SDU |
processPdu-SDU | processResponsePdu-SDU | Network-MSG

(116) sendPdu-SDU = (transportData, messageData, securityData, accessData, expectResponse)

(117) prepareOutgoingMsgIn-SDU = (transportData, messageData, securityData, accessData, expectResponse, sendPduHandle)

(118) prepareOutgoingMsgOut-SDU = (statusInformation, transportData, Network-MSG)

(119) returnResponsePdu-SDU = (messageData, securityData, accessData, maxSizeResponseScopedPdu, stateReference, statusInformation)

(120) prepareResponseMsgIn-SDU = (messageData, securityData, accessData, maxSizeResponseScopedPdu, stateReference, statusInformation)

(121) prepareResponseMsgOut-SDU = (Result, transportData, Network-MSG)

(122) prepareDataElementsIn-SDU = (transportData, Network-MSG)

(123) prepareDataElementsOut-SDU = (Result, messageData, securityData, accessData, pduType, sendPduHandle, maxSizeResponseScopedPdu, statusInformation, stateReference)

(124) processPdu-SDU = (messageData, securityData, accessData, maxSizeResponseScopedPdu, stateReference)

(125) processResponsePdu-SDU = (messageData, securityData, accessData, statusInformation, sendPduHandle)

(126) transportData = (transportDomain, transportAddress)

(127) messageData = (messageProcessingModel)

(128) securityData = (securityModel, securityName, securityLevel)

(129) accessData = (pduVersion, scopedPdu)

3.3 MSG

(130) Network-MSG = v3MPMessage | $\square$

3.4 Auxiliar PDU

(131) errorStatus = noError | genErr | commitFailed | undoFailed | wrongType | wrongLength | wrongEncoding | wrongValue | noCreation | inconsistentValue | notWritable | inconsistentName | authorizationError (*individual*)

(132) errorStatus $\leq$ Index

(133) Index $\leq$ integer

(134) VarBindList = list of VarBind^{+}

(135) VarBind = (ObjectName, ObjectValue)

(136) ObjectName $\leq$ token

(137) ObjectValue = ObjectSyntax | Exceptions

(138) ObjectSyntax = $\Box$

(139) Exceptions = unSpecified | noSuchObject | noSuchInstance | endOfMibView (*individual*)

3.5 Auxiliar SDU

(140) sendPduHandle = none | $\Box$ (*individual*)

(141) pduType = opTag

(142) statusInformation = Result = success | errorIndication

(143) success = noError

(144) errorIndication = noMPModel | noApplication | discardByMPModel | discardBySecModel | discardBySecName | discardByContextEngID | discardByContextName | discardByPduVersion | discardByRequestID | discardByOperation (*individual*)

(145) par of (E:errorIndication, V:ObjectValue) $\rightarrow$ errorIndication

3.6 Auxiliar

(146) handle $\geq$ requestId

(147) handle $\geq$ sendPduHandle

(148) handle $\geq$ stateReference

(149) agent $\geq$ Application | MPModel | Dispatcher

(150) Notification-Macro = $\Box$

(151) MaxAccess = Not-Acessible | Read-Only | Read-Create | Read-Write | $\Box$