The Network Module is a module which makes the Switching Domain functionality available to the Service Domain. All access to the Switching Domain from the Service Domain must go through this module.
Report overview list.
Chapter 1
Introduction

This document describes the design and implementation of the only OOram Module in the Network Layer of our example system. The Module represents all Switching Domain functionality seen from the example Service Domain. Since IN-Lab2 is not connected to a real network, the module contains the minimum of models needed to provide the type specification of a dummy implementation and the required Export Models.

The interface to the Switching Domain is commonly represented by sockets of different types such as sockets for connection control (leg, connection point), for data management and information collection (counters, buffers, trigger tables, etc), for status monitoring (end-user lines, queues, etc), for traffic management (filter etc), and for resource control (tone recognition devices etc).

Figure 1. IN-Lab2 support value chain.
Since our aim is to demonstrate the application of the OOram methodology rather than to create a realistic set of models, we have grossly simplified these concepts. The main reason for this is to keep the total number of objects low so that they can be visualized and animated on a computer screen. Our Network Layer module just exports two models:

1. A Manager model which acts as the client's main point of access to the network functionality. This manager serves as sockets for access load checking and access point allocation, and can provide its client with as many connection points as required.

2. A Connection Service model which provides a connection point with its legs.

The exported models describe the externally available properties of objects, their internals are encapsulated within the objects and their antecedent models are encapsulated within the module. If we were to connect our Service Domain to a real Switch Domain, the Network Layer objects may therefore be designed and implemented in terms of suitable sockets without any changes to the upper layers in our value support chain.
Chapter 2
Overview of the Network Module

A typical invocation of this module will create one instance of the Network Manager (Netw2MngrSrv), and ask it for one Connection Point, Netw2ConnPt. Then it will ask the Connection Point for two Legs (Netw2ConLeg) in order to establish a one-to-one connection in the Switching Domain. These objects and their active message paths are shown in figure 2 below, and the corresponding Use Case is shown as a monitored execution Message Sequence Chart in figure 3.

Figure 2. Objects involved in the creation of a one-to-one connection in the Switching Domain.
The objects are shown as large circles identified by the role name and the object ID. The small circles denote that messages have actually been sent along the associated path, no circle on a path means that no message sends were recorded.
Figure 3. Use Case shown as Message Sequence Chart for the creation of a one-to-one connection in the Switching Domain.

Horizontal arrows indicate that a message was sent in the given direction. A heavy arrow indicates that this message was legal according to the design models, a thin arrow indicates that this message was illegal in the sense that it was not allowed for in the design. The reason for the illegal messages in this case is that the manager is not designated as a client of the connection point.

The export/import model structure of the Network Module is shown in figure 4??.

The module contains the following models:

1. Imported models
   - The imported models would in this case be models exported from the Switching Domain. In this example, the imported models section is empty since this is a dummy module.
2. Role Models:
   - *Network Manager (Netw2MngrRM)* describes the first point of contact between a Service Domain module and the network functionality. It should not be confused by the more common network management facilities provided with most real networks.
   - *Access Point Allocation (Netw2APLRM)* describes how resources of an access point can be allocated. Notice that this model is synthesized into the manager model, making it an integral part of the management function.
   - *Access Point Load Checking (Netw2ChkRM)* describes how an access load monitor can check if an Access Point can handle new calls. Notice that this model is synthesized into the manager model, making it an integral part of the management function.
   - *Connection Model (Netw2ConnRM)* describes the creation and removal of communication paths between two or more addressable entities in the network.
   - *Use Case (Netw2MngrClient1 Use Case 1)* is the report of a monitored execution of an example of an executable specification of this module.

3. Object type specifications:
   - *Network Access Manager (Netw2MngrSPC)* specifies the types of a coordinated set of objects which play the roles of the Access Point Allocation, Access Point Monitor, Network Manager, and Connection models.
   - *Network Connection Model (Netw2ConnSPC)* specifies the types of a coordinated set of objects which play the roles of the connection model.

4. Export models. These are the only models of the module which are visible from outside the module.
   - *Network Manager (Netw2MngrEXP)* specifies how an importing model may utilize the network functionality of the Network Module.
   - *Connection Model (Netw2ConnEXP)* specifies how an importing model may utilize the connection point functionality of the Network Module.

5. Importing models. For completeness, the diagram shows the models of other modules which currently import from this module:
   - *Telephone Calling Service (Tel2ServASPC)*. The only user of our exported models is the calling (A) side of the Telephone Service.
Figure 4. Export/import structure of the Network Module models.

Plain rectangles denote role models, lines from the right edge of a rectangle means export, lines into the left edge means import. Heavy lines right and left lines in the model rectangle means that this is an object type specification, a heavy bottom line means that this model has been implemented as an executable model.

Note that a module may use its own exported models. In this example, Netw2MngrSPC imports Netw2ConnEXP.

The Export Models are described in section 3??, the Monitored Executions are described in detail in section 4??, the internal role models in section 5??, and the object type specifications in section 6??. The implementations for monitored execution corresponding to the object type specifications are given in appendix 1??.
Chapter 3
Network export models summary

3.1 Network Connection Model (Netw2ConnEXP) {Export Model}

Report automatically generated 24 July 1993

3.1.1 Area of Concern

The ConnectionControl is primarily responsible for the provision of communication paths between two or more addressable entities in the network (end user, service provider, etc). The ConnectionControl offers two different types of communication path:
1. Unidirectional communication path
2. Bidirectional communication path.

These communication paths can be built up by service primitives on the attributes: Legs and ConnectionPoints, to provide a large variety of connection configurations. But some restrictions are put on the possible combinations:
1. The parts of a communication path must all be unidirectional or bidirectional.
2. For the unidirectional case, at most one source (addressable entity) can be connected at a time (no restriction on number of connected sinks), i.e. effectively supporting point-to-multipoint configuration.
3. For the bidirectional case, at most two addressable entities may be connected at a time, i.e. effectively supporting point-to-point configurations.

The objects of this role model are abstract, in the sense that they are responsible for the isolation of the Client from the ATM specific network technology.
3.1.2 The Roles

1. **Net2Connection Client** (*Netw2ConnClient*) *(Type)*. The service instance, *Netw2ConnClient*, views the network as a virtual switch object and represents a user of the Connection Control.

2. **Net2Connection Leg** (*Netw2ConnLeg*) *(Type)*. This role represents a communication path from a connection point towards an addressable entity in the network (e.g. an end user terminal). A Leg is always associated with a ConnectionPoint. Leg is responsible for storing the information characterizing the leg.

3. **Net2Connection Point** (*Netw2ConnPoint*) *(Type)*. This role represents an interconnection or logical bridge allowing information to flow between legs.
3.1.3 Interaction Scenarios

3.2 Network Access Manager EXP (Netw2MngrEXP) {Export Model}

Report automatically generated 24 July 1993

3.2.1 Area of Concern

This role model describes the main access point to the Switching Domain. Its purpose is to provide the client with the available Switching Domain services. It combines the functionality of a general switch manager and some socket types.

The ConnectionControl is primarily responsible for the provision of communication paths between two or more addressable entities in the network (end user, service provider, etc). The
ConnectionControl offers two different types of communication path:
1. Unidirectional communication path
2. Bidirectional communication path.

These communication paths can be built up by service primitives on the attributes: Legs and ConnectionPoints, to provide a large variety of connection configurations. But some restrictions are put on the possible combinations:
1. The parts of a communication path must all be unidirectional or bidirectional.
2. For the unidirectional case, at most one source (addressable entity) can be connected at a time (no restriction on number of connected sinks), i.e. effectively supporting point-to-multipoint configuration.
3. For the bidirectional case, at most two addressable entities may be connected at a time, i.e. effectively supporting point-to-point configurations.

The objects of this role model are abstract, in the sense that they are responsible for the isolation of the Client from the ATM specific network technology.

### 3.2.2 The Roles

1. **Network Manager Client** *(Netw2MngrClient)* *(Type)*. Any Service Domain object which has access to the Switching Domain services.

2. **Network Access Manager** *(Netw2Mngr)* *(Type)*. This object is a main access point to the Switching domain, giving access to the specialized Switching Domain services.

   The service client views the network as a virtual switch object and represents a user of the Connection Control.
Chapter 4
Detailed report of the monitored execution

4.1 Netw2MngrClient1 Use Case 1 {Role Model}


4.1.1 Area of Concern

This example shows a typical message sequence when establishing and freeing a plain telephone connection.

Stimulus messages
1. Netw2MngrClnt#106>>openSocketFor:responseLevel:
2. Netw2MngrClnt#106>>createCp:with:for:
3. Netw2MngrClnt#106>>createLeg:direction:to:qoc:
4. Netw2MngrClnt#106>>freeCp
4.1.2 The Roles

1. Netw2MngrClient#106 (Netw2MngrClnt#106) \{Role\}.
2. Netw2MngrServer#107 (Netw2MngrSrvr#107) \{Role\}.
3. Netw2ConnPoint#108 (Netw2ConnPt#108) \{Role\}.
4. Netw2ConnLeg#109 \{Role\}.
5. Netw2ConnLeg#110 \{Role\}.
4.1.3 Interaction Scenarios

Figure 6. Netw2MngrClient1 example1 {Scenario}

4.1.4 Message Sets

1. Netw2MngrClient#106 (Netw2MngrClnt#106) {Role}
   - 1 {Port}.
     - executed {Contract}.
       - openSocketFor:responseLevel: {Message}.
       - createCp:with:for: {Message}.
   - 2 {Port}.
     - executed {Contract}.
       - createLeg:direction:to:qoc: {Message}.
       - freeCp {Message}. 
2. **Netw2MngrServer#107 (Netw2MngrSrvr#107) {Role}**
   - 3 {Port}.
     - executed {Contract}.
     - openCPFor:type:responseLevel: {Message}.

3. **Netw2ConnPoint#108 (Netw2ConnPt#108) {Role}**
   - 4 {Port}.
     - executed {Contract}.
     - connPt:client:direction:to:qoc: {Message}.
     - freeLeg {Message}.
   - 5 {Port}.
     - executed {Contract}.
     - connPt:client:direction:to:qoc: {Message}.
     - freeLeg {Message}. 
Chapter 5
Role Models of the Network Module

5.1 Network Connection Model (Netw2ConnRM) {Role Model}

Report including the following selections automatically generated 23 July 1993: Role Models
Area of Concern - stimuli Role list - diagram -role explanation - role import map Scenarios -
explanation Message Sets -port explanation - contract explanation -message explanation FSM
Implementation

5.1.1 Area of Concern

The ConnectionControl is primarily responsible for the provision of communication paths
between two or more addressable entities in the network (end user, service provider, etc). The
ConnectionControl offers two different types of communication path:
1. Unidirectional communication path
2. Bidirectional communication path.

These communication paths can be built up by service primitives on the attributes: Legs and
ConnectionPoints, to provide a large variety of connection configurations. But some restrictions
are put on the possible combinations:
1. The parts of a communication path must all be unidirectional or bidirectional.
2. For the unidirectional case, at most one source (addressable entity) can be connected at a time
(no restriction on number of connected sinks), i.e. effectively supporting point-to-multipoint
configuration.
3. For the bidirectional case, at most two addressable entities may be connected at a time, i.e.
effectively supporting point-to-point configurations.

The objects of this role model are abstract, in the sense that they are responsible for the isolation
of the Client from the ATM specific network technology.
5.1.2 The Roles

1. **Netw2Connection Client (Netw2ConnClient) {Role}.** The service instance, Netw2ConnClient, views the network as a virtual switch object and represents a user of the Connection Control.

2. **Netw2Connection Leg (Netw2ConnLeg) {Role}.** This role represents a communication path from a connection point towards an addressable entity in the network (e.g. an end user terminal). A Leg is always associated with a ConnectionPoint. Leg is responsible for storing the information characterizing the leg.

3. **Netw2Connection Point (Netw2ConnPoint) {Role}.** This role represents an interconnection or logical bridge allowing information to flow between legs.
5.1.3 Interaction Scenarios

![Interaction Scenarios Diagram]

*Figure 7. Rudimentary create and free sequence. {Scenario}* 

5.1.4 Message Sets

1. **Net2Connection Client (Netw2ConnClient) {Role}**
- **connectionPoint (cp) {Port}**.
  - **ConnPoint<ConnClient {Contract}**.
    - **openCPFor: aClient type: typ responseLevel: aSymbol {Message}**. Requests an initiation of a ConnectionPoint of the connection control service to start the session. This message must be the first to be received by the ConnectionPoint and cannot be repeated.
      
      *type* is {#Simplex, #Duplex}
      
      *responseLevel* is one of {#OnlyNack, #AckNack}.
    - **freeCp {Message}**. This message is used by the ConnClient (service logic) to request a connection point with all its legs and all associated resources to be released.
  - **createLeg: type direction: dir to: destID qoc: qoc {Message}**. Requests the creation of a leg to an addressable entity.
      
      *type* is {#Simplex, #Duplex}
      
      *dir* is {#Upstream, #Downstream, #None}
      
      *destID* is accessPointID
      
      *qoc* is Integer representing required bandwidth.
    - **freeLeg: aLeg {Message}**. Requests that a leg with all its associated resources to be released.
  - **legs (leg) {Port}**
    - **Leg<ConnClient {Contract}**.
      - **freeLeg {Message}**. This message is used to request a leg and all associated resources to be released.
      - **modifyQoc: qoc {Message}**. Requests modification of the Quality Of Connection of the Leg. *qoc* is Integer representing required bandwidth.
      - **modifyDirection: dir {Message}**. Requests modification of the Direction of Communication of the Leg. *dir* is {#Upstream, #Downstream, #None}

2. **Netw2Connection Leg (Netw2ConnLeg) {Role}**
   - **connectionPoint (cp) {Port}**.
     - **ConnPoint<Leg {Contract}**.
       - **legFreed: aLeg {Message}**. The given Leg has been freed and should be removed from the receiver.
3. **Netw2Connection Point (Netw2ConnPoint) \{Role\}**

   - **legs (lg) \{Port\}**.
   - **Leg<ConnPoint \{Contract\}**.
     - **connPt: aConnPoint client: aClient direction: dir to: swID qoc: qoc \{Message\}.**
       Initialize with the given attributes.
       - \textit{dir} is \{#Upstream, #Downstream, #None\}
       - \textit{swID} is accessPointID
       - \textit{qoc} is Integer representing required bandwidth.
   - **Leg<ConnClient \{Contract\}**.
     - **freeLeg \{Message\}.** This message is used to request a leg and all associated resources to be released.
     - **modifyQoc: qoc \{Message\}.** Requests modification of the Quality Of Connection of the Leg. \textit{qoc} is Integer representing required bandwidth.
     - **modifyDirection: dir \{Message\}.** Requests modification of the Direction of Communication of the Leg. \textit{dir} is \{#Upstream, #Downstream, #None\}

5.2 **Network Access Load Checking (Netw2ChkRM) \{Role Model\}**

Report including the following selections automatically generated 23 July 1993: Role Models
Area of Concern - stimuli Role list - diagram -role explanation - role import map Scenarios - explanation Message Sets -port explanation - contract explanation -message explanation FSM Implementation

5.2.1 **Area of Concern**

This role model shows how an access load monitor can check if an access point can handle new calls.

*Stimulus messages*

1. **Netw2ALM\>loadCheck:for:** This message can be sent to an Access Point to evaluate if it can accept a new call with a certain load (anInteger) of a given type (aSymbol). Return \textit{true} iff load can be accepted, otherwise \textit{false}. Note that this message does not allocate any resources in the Access Point, it just checks if the Access Point could accept the load.
5.2.2 The Roles

1. **Network Access Load Monitor (Netw2ALM) {Role}**. This role shall be able to check whether an access point can accept new calls of a given type and with a given load.

2. **Network Access Point (Netw2AP) {Role}**. This role represents the lower OSI layers of a terminal-network interface. The exact nature of the terminal-network interface can vary, e.g. it could correspond to a mobile access with a mobile telephone, an analog access with a DTMF telephone or an ISDN access with an ISDN telephone. It is assumed that the terminal-network interface has a finite amount of resources for new calls.

5.2.3 Message Sets

1. **Network Access Load Monitor (Netw2ALM) {Role}**
   - accessPoint (ap) {Port}.
   - AP<ALM {Contract}.
   - loadCheck: type for: load {Message}. This message can be sent to an Access Point to evaluate if it can accept a new call with a certain load (anInteger) of a given type (aSymbol). Return true iff load can be accepted, otherwise false. Note that this message does not allocate any resources in the Access Point, it just checks if the Access Point could accept the load.

5.3 Network Access Point Allocation (Netw2APLRM) {Role Model}
5.3.1 Area of Concern

This role model shows how resources of an Access Point can be allocated.

Stimulus messages

1. **Netw2A>>deallocate:load**: This message allows the Allocator to release resources in the AccessPoint for a certain load (anInteger) of a given type (aSymbol). Return `true` iff load can be accepted, otherwise `false`.

2. **Netw2A>>allocate:load**: This message allows the Allocator to allocate resources in the AccessPoint for a new call with a certain load (anInteger) of a given type (aSymbol). Return `true` iff load can be accepted, otherwise `false`.

5.3.2 The Roles

1. **Network Access Point Allocator (Netw2A) [Role]**. This role represents an object which has the ability to allocate resources in an access point.

2. **Network AccessPoint (Netw2AP) [Role]**. This role represents the lower OSI layers of a terminal-network interface. The exact nature of the terminal-network interface can vary, e.g. it could correspond to a mobile access with a mobile telephone, an analog access with a DTMF telephone or an ISDN access with an ISDN telephone. It is assumed that the terminal-network interface has a finite amount of resources for new calls. The allocator can allocate and release resources in the terminal-network interface.
5.3.3 Message Sets

1. **Network Access Point Allocator (Netw2A) {Role}**
   - **accessPoint (ap) {Port}**.
   - **AP<A {Contract}**.
     - **allocate: type load: load {Message}**. This message allows the Allocator to allocate resources in the AccessPoint for a new call with a certain load (anInteger) of a given type (aSymbol). Return true iff load can be accepted, otherwise false.
     - **deallocate: type load: load {Message}**. This message allows the Allocator to release resources in the AccessPoint for a certain load (anInteger) of a given type (aSymbol). Return true iff load can be accepted, otherwise false.

5.4 **Network Access Manager (Netw2MngrRM) {Role Model}**

Report including the following selections automatically generated 23 July 1993: Role Models Area of Concern - stimuli Role list - diagram - role explanation - role import map Scenarios - explanation Message Sets - port explanation - contract explanation - message explanation FSM Implementation

5.4.1 **Area of Concern**

This role model describes the main access point to the Switching Domain. Its purpose is to provide the client with the available Switching Domain services. It combines the functionality of a general switch manager and some socket types.

**Stimulus messages**

1. **Netw2MngrClient>>deallocate:load**: This message allows the Allocator to release resources in the AccessPoint for a certain load (anInteger) of a given type (aSymbol). Return true iff load can be accepted, otherwise false.

2. **Netw2MngrClient>>loadCheck:for**: This message can be sent to an Access Point to evaluate if it can accept a new call with a certain load (anInteger) of a given type (aSymbol). Return true iff load can be accepted, otherwise false. Note that this message does not allocate any resources in the Access Point, it just checks if the Access Point could accept the load.
3. `Netw2MngrClient>>allocate:load:` This message allows the Allocator to allocate resources in the AccessPoint for a new call with a certain load (aInteger) of a given type (aSymbol). Return `true` iff load can be accepted, otherwise `false`.

4. `Netw2MngrClient>>freeCp:` This message is used by the SockClient (service logic) to release a ConnectionPoint with all its legs and all its associated resources.

5. `Netw2MngrClient>>createCp:with:for:` Create a ConnectionPoint. The legs must all be of the same `type`.
   - `type` {#Simplex, #Duplex}.
   - `respLev` is one of {#OnlyNack, #AckNack}.

### 5.4.2 The Roles

1. **Network Manager Client** (`Netw2MngrClient`) **(Role)**. Any Service Domain object which has access to the Switching Domain services.

2. **Network Access Manager** (`Netw2Mngr`) **(Role)**. This object is a main access point to the Switching domain, giving access to the specialized Switching Domain services.

```
<table>
<thead>
<tr>
<th>Netw2MngrRM</th>
<th>Netw2APLRM</th>
<th>Netw2ChkRM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Netw2MngrClient</td>
<td>Netw2A</td>
<td>Netw2ALM</td>
</tr>
<tr>
<td>Netw2Mngr</td>
<td>Netw2AP</td>
<td>Netw2AP</td>
</tr>
</tbody>
</table>
```

*Table 8. Import relationships for Network Access Manager ([RMRoleModel](#)).*
5.4.3 Message Sets

1. Network Manager Client (Netw2MngrClient) \{Role\}
   - switchManager (mng) \{Port\}.
   - MngrServ<MngrClient \{Contract\}.
     - openSocketFor: client responseLevel: lev \{Message\}. Requests the initialization of a Socket. level is one of \{#OnlyNack #AckNack\} This message must be the first message received by the Socket object, and cannot be received at any other time.
     - createCp: type with: respLev for: aClient \{Message\}. Create a ConnectionPoint. The legs must all be of the same type.
       - type \{#Simplex, #Duplex\}.
       - respLev is one of \{#OnlyNack, #AckNack\}.
     - freeCp: aCP \{Message\}. This message is used by the SockClient (service logic) to release a ConnectionPoint with all its legs and all its associated resources.
Chapter 6
Object type specifications in the Network Module

6.1 Network Connection Model (Netw2ConnSPC) {Type Specification}

Report including the following selections automatically generated 23 July 1993: Type Specifications Area of Concern - stimuli Role list - diagram -role explanation - role import map Scenarios - explanation Message Sets -port explanation - contract explanation -message explanation FSM

6.1.1 Area of Concern

The ConnectionControl is primarily responsible for the provision of communication paths between two or more addressable entities in the network (end user, service provider, etc). The ConnectionControl offers two different types of communication path:
1. Unidirectional communication path
2. Bidirectional communication path.

These communication paths can be built up by service primitives on the attributes: Legs and ConnectionPoints, to provide a large variety of connection configurations. But some restrictions are put on the possible combinations:
1. The parts of a communication path must all be unidirectional or bidirectional.
2. For the unidirectional case, at most one source (addressable entity) can be connected at a time (no restriction on number of connected sinks), i.e. effectively supporting point-to-multipoint configuration.
3. For the bidirectional case, at most two addressable entities may be connected at a time, i.e. effectively supporting point-to-point configurations.

The objects of this role model are abstract, in the sense that they are responsible for the isolation of the Client from the ATM specific network technology.
6.1.2 The Roles

1. **Net2Connection Client** *(Netw2ConnClient) {Type}* The service instance, Netw2ConnClient, views the network as a virtual switch object and represents a user of the Connection Control.

2. **Netw2Connection Leg** *(Netw2ConnLeg) {Type}* This role represents a communication path from a connection point towards an addressable entity in the network (e.g. an end user terminal). A Leg is always associated with a ConnectionPoint. Leg is responsible for storing the information characterizing the leg.

3. **Netw2Connection Point** *(Netw2ConnPoint) {Type}* This role represents an interconnection or logical bridge allowing information to flow between legs.

<table>
<thead>
<tr>
<th><code>Netw2ConnSPC</code></th>
<th><code>Netw2ConnRM</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Netw2ConnClient</td>
<td>Netw2ConnClient</td>
</tr>
<tr>
<td>Netw2ConnLeg</td>
<td>Netw2ConnLeg</td>
</tr>
<tr>
<td>Netw2ConnPoint</td>
<td>Netw2ConnPoint</td>
</tr>
</tbody>
</table>

*Table 9. Import relationships for Network Connection Model {RMTTypeSpecification}.*
6.2 Network Access Manager SPC (Netw2MngrSPC) {Type Specification}

Report including the following selections automatically generated 23 July 1993: Type Specifications Area of Concern - stimuli Role list - diagram -role explanation - role import map Scenarios - explanation Message Sets -port explanation - contract explanation -message explanation FSM

6.2.1 Area of Concern

This role model describes the main access point to the Switching Domain. Its purpose is to provide the client with the available Switching Domain services. It combines the functionality of a general switch manager and some socket types.

The ConnectionControl is primarily responsible for the provision of communication paths between two or more addressable entities in the network (end user, service provider, etc). The ConnectionControl offers two different types of communication path:
1. Unidirectional communication path
2. Bidirectional communication path.

These communication paths can be built up by service primitives on the attributes: Legs and ConnectionPoints, to provide a large variety of connection configurations. But some restrictions are put on the possible combinations:
1. The parts of a communication path must all be unidirectional or bidirectional.
2. For the unidirectional case, at most one source (addressable entity) can be connected at a time (no restriction on number of connected sinks), i.e. effectively supporting point-to-multipoint configuration.
3. For the bidirectional case, at most two addressable entities may be connected at a time, i.e. effectively supporting point-to-point configurations.

The objects of this role model are abstract, in the sense that they are responsible for the isolation of the Client from the ATM specific network technology.
6.2.2 The Roles

1. **Network Manager Client (Netw2MngrClient) (Type)**. Any Service Domain object which has access to the Switching Domain services.

2. **Network Access Manager (Netw2Mngr) (Type)**. This object is a main access point to the Switching domain, giving access to the specialized Switching Domain services.

   The service instance, Netw2ConnClient, views the network as a virtual switch object and represents a user of the Connection Control.

3. **Netw2Connection Leg (Netw2ConnLeg) (Type)**. This role represents a communication path from a connection point towards an addressable entity in the network (e.g. an end user terminal). A Leg is always associated with a ConnectionPoint. Leg is responsible for storing the information characterizing the leg.

4. **Netw2Connection Point (Netw2ConnPoint) (Type)**. This role represents an interconnection or logical bridge allowing information to flow between legs.
<table>
<thead>
<tr>
<th>Netw2MngrSPC</th>
<th>Netw2MngrRM</th>
<th>Netw2ConnEXP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Netw2MngrClient</td>
<td>Netw2MngrClient</td>
<td></td>
</tr>
<tr>
<td>Netw2Mngr</td>
<td>Netw2Mngr</td>
<td>Netw2ConnClient</td>
</tr>
<tr>
<td>Netw2ConnLeg</td>
<td></td>
<td>Netw2ConnLeg</td>
</tr>
<tr>
<td>Netw2ConnPoint</td>
<td></td>
<td>Netw2ConnPoint</td>
</tr>
</tbody>
</table>

*Table 10. Import relationships for Network Access Manager SPC {RMTypeSpecification}.*
Chapter 7
Network export models

7.1 Network Connection Model (Netw2ConnEXP) {Export Model}  

Report including the following selections automatically generated 23 July 1993: Export Models  
Area of Concern - stimuli Role list - diagram -role explanation Scenarios - explanation  
Message Sets -imports - port explanation - contract explanation -message explanation FSM  
Implementation

7.1.1 Area of Concern

The ConnectionControl is primarily responsible for the provision of communication paths  
between two or more addressable entities in the network (end user, service provider, etc). The  
ConnectionControl offers two different types of communication path:  
1. Unidirectional communication path  
2. Bidirectional communication path.

These communication paths can be built up by service primitives on the attributes: Legs and  
ConnectionPoints, to provide a large variety of connection configurations. But some restrictions  
are put on the possible combinations:  
1. The parts of a communication path must all be unidirectional or bidirectional.  
2. For the unidirectional case, at most one source (addressable entity) can be connected at a time  
(no restriction on number of connected sinks), i.e. effectively supporting point-to-multipoint  
configuration.  
3. For the bidirectional case, at most two addressable entities may be connected at a time, i.e.  
effectively supporting point-to-point configurations.

The objects of this role model are abstract, in the sense that they are responsible for the isolation  
of the Client from the ATM specific network technology.
7.1.2 The Roles

1. **Netw2Connection Client (Netw2ConnClient) \(\{\text{Type}\}\).** The service instance, Netw2ConnClient, views the network as a virtual switch object and represents a user of the Connection Control.

2. **Netw2Connection Leg (Netw2ConnLeg) \(\{\text{Type}\}\).** This role represents a communication path from a connection point towards an addressable entity in the network (e.g. an end user terminal). A Leg is always associated with a ConnectionPoint. Leg is responsible for storing the information characterizing the leg.

3. **Netw2Connection Point (Netw2ConnPoint) \(\{\text{Type}\}\).** This role represents an interconnection or logical bridge allowing information to flow between legs.
7.1.3 Interaction Scenarios

![Diagram of interaction scenarios](Image)

**Figure 11. Rudimentary create and free sequence. (Scenario)**

7.1.4 Message Sets

1. **Net2Connection Client (Netw2ConnClient) {Type}**
   - legs (leg) {Port}
     - Leg<ConnClient {Contract}.
     - **freeLeg {Message}**. This message is used to request a leg and all associated resources to be released.
     - **modifyQoc: qoc {Message}**. Requests modification of the Quality Of Connection of the Leg. *qoc* is Integer representing required bandwidth.
     - **modifyDirection: dir {Message}**. Requests modification of the Direction of Communication of the Leg. *dir* is \{#Upstream, #Downstream, #None\}
- **connectionPoint (cp) {Port}**.
- **ConnPoint<ConnClient {Contract}**.
  - **openCPFor: aClient type: typ responseLevel: aSymbol {Message}**. Requests an initiation of a ConnectionPoint of the connection control service to start the session. This message must be the first to be received by the ConnectionPoint and cannot be repeated.  
    - type is {#Simplex, #Duplex}  
    - responseLevel is one of {#OnlyNack, #AckNack}.
- **freeCp {Message}**. This message is used by the ConnClient (service logic) to request a connection point with all its legs and all associated resources to be released.
- **createLeg: type direction: dir to: destID qoc: qoc {Message}**. Requests the creation of a leg to an addressable entity.
  - type is {#Simplex, #Duplex}  
  - dir is {#Upstream, #Downstream, #None}  
  - destID is accessPointID  
  - qoc is Integer representing required bandwidth.
- **freeLeg: aLeg {Message}**. Requests that a leg with all its associated resources to be released.

2. **Netw2Connection Leg (Netw2ConnLeg) {Type}**
   - **connectionPoint (cp) {Port}**.
     - **ConnPoint<Leg {Contract}**.
       - **legFreed: aLeg {Message}**. The given Leg has been freed and should be removed from the receiver.

3. **Netw2Connection Point (Netw2ConnPoint) {Type}**
   - **legs (lg) {Port}**.
     - **Leg<ConnPoint {Contract}**.
       - **connPt: aConnPoint client: aClient direction: dir to: swID qoc: qoc {Message}**. Initialize with the given attributes.
         - dir is {#Upstream, #Downstream, #None}  
         - swID is accessPointID  
         - qoc is Integer representing required bandwidth.
     - **Leg<ConnClient {Contract}**.
       - **freeLeg {Message}**. This message is used to request a leg and all associated resources to be released.
       - **modifyQoc: qoc {Message}**. Requests modification of the Quality Of Connection of the Leg. qoc is Integer representing required bandwidth.
       - **modifyDirection: dir {Message}**. Requests modification of the Direction of Communication of the Leg. dir is {#Upstream, #Downstream, #None}.
7.2 Network Access Manager EXP (Netw2MngrEXP) {Export Model}

Report including the following selections automatically generated 23 July 1993: Export Models
Area of Concern - stimuli Role list - diagram -role explanation Scenarios - explanation
Message Sets -imports - port explanation - contract explanation -message explanation FSM
Implementation

7.2.1 Area of Concern

This role model describes the main access point to the Switching Domain. Its purpose is to provide the client with the available Switching Domain services. It combines the functionality of a general switch manager and some socket types.

The ConnectionControl is primarily responsible for the provision of communication paths between two or more addressable entities in the network (end user, service provider, etc). The ConnectionControl offers two different types of communication path:
1. Unidirectional communication path
2. Bidirectional communication path.

These communication paths can be built up by service primitives on the attributes: Legs and ConnectionPoints, to provide a large variety of connection configurations. But some restrictions are put on the possible combinations:
1. The parts of a communication path must all be unidirectional or bidirectional.
2. For the unidirectional case, at most one source (addressable entity) can be connected at a time (no restriction on number of connected sinks), i.e. effectively supporting point-to-multipoint configuration.
3. For the bidirectional case, at most two addressable entities may be connected at a time, i.e. effectively supporting point-to-point configurations.

The objects of this role model are abstract, in the sense that they are responsible for the isolation of the Client from the ATM specific network technology.
7.2.2 The Roles

1. **Network Manager Client (Netw2MngrClient)** *(Type)*. Any Service Domain object which has access to the Switching Domain services.

2. **Network Access Manager (Netw2Mngr)** *(Type)*. This object is a main access point to the Switching domain, giving access to the specialized Switching Domain services.

The service client views the network as a virtual switch object and represents a user of the Connection Control.

7.2.3 Message Sets

1. **Network Manager Client (Netw2MngrClient)** *(Type)*

   - switchManager *(mng)* *(Port)*.

   - MngrServ<MngrClient *(Contract)*.

     - openSocketFor: client responseLevel: lev *(Message)*. Requests the initialization of a Socket. *level* is one of {#OnlyNack #AckNack} This message must be the first message received by the Socket object, and cannot be received at any other time.

     - createCp: type with: respLev for: aClient *(Message)*. Create a ConnectionPoint. The legs must all be of the same *type*.
       
       *type* {#Simplex, #Duplex}.

       *respLev* is one of {#OnlyNack, #AckNack}.

     - freeCp: aCP *(Message)*. This message is used by the SockClient (service logic) to release a ConnectionPoint with all its legs and all its associated resources.

   - AP<A *(Contract)*.

     - allocate: type load: load *(Message)*. This message allows the Allocator to allocate resources in the AccessPoint for a new call with a certain load (anInteger) of a given type (aSymbol). Return *true* iff load can be accepted, otherwise *false*.

     - deallocate: type load: load *(Message)*. This message allows the Allocator to release resources in the AccessPoint for a certain load (anInteger) of a given type (aSymbol). Return *true* iff load can be accepted, otherwise *false*. 
- **AP<ALM (Contract).**

- `loadCheck: type for: load` *(Message)*. This message can be sent to an Access Point to evaluate if it can accept a new call with a certain load (an Integer) of a given type (a Symbol). Return `true` iff load can be accepted, otherwise `false`. Note that this message does not allocate any resources in the Access Point, it just checks if the Access Point could accept the load.
Appendix 1
Implementation specifications for monitored execution

App 1.1 Network Connection Model (Netw2ConnSPC) {Type Specification}

Report including the following selections automatically generated 23 July 1993: Type Specifications Implementation

App 1.1.1 Network Connection Model {Implementation}

RMSObject1 subclass: #Netw2ConnClient0
  instanceVariableNames: 'connectionPoint legs '
  classVariableNames: "
  poolDictionaries: "
  category: 'ZZZInLab-Netw2+'

The service instance, Netw2ConnClient, views the network as a virtual switch object and represents a user of the Connection Control.

Instance variables:
  connectionPoint<Netw2ConnPoint0>
  legs <Netw2ConnLeg0>  No Explanation

Subclasses should not implement methods for:

Subclasses must implement the following messages:

RMSObject1 subclass: #Netw2ConnLeg0
  instanceVariableNames: 'client connectionPoint targetSwitchID direction qualityOfConnection '
  classVariableNames: "
  poolDictionaries: "
  category: 'ZZZInLab-Netw2+'

This role represents a communication path from a connection point towards an addressable entity in the network (e.g. an end user terminal). A Leg is always associated with a ConnectionPoint. Leg is responsible for storing the information characterizing the leg.

Instance variables:
  client<Netw2ConnClient0>
  connectionPoint<Netw2ConnPoint0>
  targetSwitchID <Object>  Unknown type
  direction <Object>  Unknown type
  qualityOfConnection <Object>  Unknown type
Subclasses should not implement methods for:

Subclasses must implement the following messages:

Leg<ConnPoint
  connPt:client:direction:to:qoc:
Leg<ConnClient
  freeLeg
  modifyDirection:
  modifyQoc:

RMSObject1 subclass: #Netw2ConnPoint0
  instanceVariableNames: 'connectionUser legs responseLevel type '
  classVariableNames: ''
  poolDictionaries: ''
  category: 'ZZZInLab-Netw2+'

This role represents an interconnection or logical bridge allowing information to flow between legs.

Instance variables:
  connectionUser <Netw2ConnClient0>
  legs <Netw2ConnLeg0>
  responseLevel <Object> Unknown type
  type <Object> Unknown type

Subclasses should not implement methods for:

Subclasses must implement the following messages:

ConnPoint<ConnClient
  createLeg:direction:to:qoc:
  freeCp
  freeLeg:
  openCPFor:type:responseLevel:
ConnPoint<Leg
  legFreed:

Netw2ConnClient0 subclass: #Netw2ConnClient1
  instanceVariableNames: ''
  classVariableNames: ''
  poolDictionaries: ''
  category: 'ZZZInLab-Netw2+'

Netw2ConnLeg0 subclass: #Netw2ConnLeg1
  instanceVariableNames: ''
  classVariableNames: ''
  poolDictionaries: ''
  category: 'ZZZInLab-Netw2+'

Netw2ConnPoint0 subclass: #Netw2ConnPoint1
  instanceVariableNames: ''
  classVariableNames: ''
  poolDictionaries: ''
  category: 'ZZZInLab-Netw2+'

Netw2ConnLeg1
class>>subclass:instanceVariableNames:classVariableNames:poolDictionaries:category: (compilation)
subclass: t instanceVariableNames: f classVariableNames: d poolDictionaries: s category: cat
  " Automatically generated."
TMWarningDialog
  warning: 'This class is immutable and cannot be subclassed.'.
^nil

Netw2ConnPoint1
class>>subclass:instanceVariableNames:classVariableNames:poolDictionaries:category: (compilation)
subclass: t instanceVariableNames: f classVariableNames: d poolDictionaries: s category: cat
  " Automatically generated."
TMWarningDialog
  warning: 'This class is immutable and cannot be subclassed.'.
^nil

Netw2ConnLeg1>>connPt:client:direction:to:qoc: (Leg<ConnPoint)
connPt: aConnPoint client: aClient direction: dir to: swID qoc: qoc
  " Initialize with the given attributes."
  " dir is {#Upstream, #Downstream, #None}"
  " swID is accessPointID"
  " qoc is Integer representing required bandwidth."
client := aClient.
connectionPoint := aConnPoint.
qualityOfConnection := qoc.
targetSwitchID := swID.
direction := dir.

Netw2ConnLeg1>>freeLeg (Leg<ConnClient)
freeLeg
  " This message is used to request a leg and all associated resources to be released."
client := nil.
connectionPoint notNil ifTrue: [connectionPoint legFreed: self guard].
connectionPoint := nil.
qualityOfConnection := nil.
targetSwitchID := nil.
direction := nil.

Netw2ConnLeg1>>modifyDirection: (Leg<ConnClient)
modifyDirection: dir
  " Requests modification of the Direction of Communication of the Leg."
  " dir is {#Upstream, #Downstream, #None}"
direction := dir.

Netw2ConnLeg1>>modifyQoc: (Leg<ConnClient)
modifyQoc: qoc
  " Requests modification of the Quality Of Connection of the Leg. qoc is Integer representing required bandwidth."
qualityOfConnection := qoc.

Netw2ConnPoint1>>createLeg:direction:to:qoc: (ConnPoint<ConnClient)
createLeg: tp direction: dir to: destID qoc: qoc
  " Requests the creation of a leg to an addressable entity."
  " type is {#Simplex, #Duplex}"
  " dir is {#Upstream, #Downstream, #None}"
  " destID is accessPointID"
  " qoc is Integer representing required bandwidth."
  | leg |
  ifTrue: [legs := RMSOrderedCollection new].
tp ~= type ifTrue: [self dpsCaution: 'ConnType=', type, ' legType=', tp, ' Should be same!!'].

legs add: leg.

\^leg

Netw2ConnPoint1>>freeCp (ConnPoint<ConnClient)
freeCp
" This message is used by the ConnClient (service logic) to request "
" a connection point with all its legs and all associated resources "
" to be released. "
| a |
connectionUser := nil.
a := 0.
legs copy do: [:leg | a := a + 1. leg freeLeg].
responseLevel := nil.
type := nil.

Netw2ConnPoint1>>freeLeg: (ConnPoint<ConnClient)
freeLeg: aLeg
" Requests that a leg with all its associated resources to be released. "
aLeg freeLeg.

Netw2ConnPoint1>>openCPFor:type:responseLevel: (ConnPoint<ConnClient)
openCPFor: aClient type: typ responseLevel: aSymbol
" Requests an initiation of a ConnectionPoint of the connection control "
" service to start the session. This message must be the first to be "
" received by the ConnectionPoint and cannot be repeated. "
" type is {#Simplex, #Duplex} "
" responseLevel is one of {#OnlyNack, #AckNack}. "
connectionUser := aClient.
legs := RMSOrderedCollection new.
responseLevel := aSymbol.
type := typ.

Netw2ConnPoint1>>legFreed: (ConnPoint<Leg)
legFreed: aLeg
" No Explanation "
| found newColl |
legs isNil ifTrue: [legs := RMSOrderedCollection new].
found := false.
newColl := RMSOrderedCollection new: legs size.
legs do:
[\^leg | aLeg == leg ifTrue: [found := true] ifFalse: [newColl add: leg]].
legs := newColl.
found ifFalse: [self dpsCaution: aLeg printString, ' not in this CP.'].

App 1.2 Network Access Manager SPC (Netw2MngrSPC) {Type Specification}

Report including the following selections automatically generated 23 July 1993: Type Specifications Implementation
App 1.2.1 Network Access Manager SPC {Implementation}

RMSObject1 subclass: #Netw2MngrClient0
  instanceVariableNames: 'switchManager'
  classVariableNames: '
  poolDictionaries: ''
  category: 'ZZZInLab-Netw2+'

Any Service Domain object which has access to the Switching Domain services.

Instance variables:
  switchManager <Netw2MngrServer0>

Subclasses should not implement methods for:

Subclasses must implement the following messages:

RMSObject1 subclass: #Netw2MngrServer0
  instanceVariableNames: 'connectionPoint legs'
  classVariableNames: ''
  poolDictionaries: ''
  category: 'ZZZInLab-Netw2+'

This object is a main access point to the Switching domain, giving access to the specialized Switching Domain services. The service instance, Netw2ConnClient, views the network as a virtual switch object and represents a user of the Connection Control.

Instance variables:
  connectionPoint <RMSObject1>
  legs <RMSObject1> No Explanation

Subclasses should not implement methods for:

Subclasses must implement the following messages:
  MngrServ<MngrClient
  createCp:with:for:
  freeCp:
  openSocketFor:responseLevel:
  AP<A
  allocate:load:
  deallocate:load:
  AP<ALM
  loadCheck:for:

Netw2MngrClient0 subclass: #Netw2MngrClient1
  instanceVariableNames: ''
  classVariableNames: ''
  poolDictionaries: ''
  category: 'ZZZInLab-Netw2+'

Netw2MngrServer0 subclass: #Netw2MngrServer1
  instanceVariableNames: ''
  classVariableNames: 'Default Previous'
  poolDictionaries: ''
  category: 'ZZZInLab-Netw2+'
Netw2MngrClient1 class>>example1 (examples)
example1
  RMSGuard1 startTrace: 'Netw2MngrClient1 example1'.
  self new example1.
  RMSGuard1 endAndInspectTrace.

Netw2MngrServer1 class>>default (instance creation)
default
  " Netw2MngrServer1 default"
  Default isNil ifTrue: [Default := self new].
  ^Default

Netw2MngrServer1 class>>subclass:instanceVariableNames:classVariableNames:poolDictionaries:category: (compilation)
subclass: t instanceVariableNames: f classVariableNames: d poolDictionaries: s category: cat
  " Automatically generated. "
  TMWarningDialog
    warning: 'This class is immutable and cannot be subclassed.'.
    ^nil

Netw2MngrServer1 class>>initialize (class initialization)
initialize
  Default := nil.
  Previous := false.

Netw2MngrClient1>>example1 (private)
exmaple1
  | serv connectionPoint legs |
  serv := Netw2MngrServer1 new.
  serv openSocketFor: self responseLevel: #OnlyNack.
  legs := RMSOrderedCollection new.
  legs addLast: (connectionPoint createLeg: #Simplex direction: #Upstream to: '1234' qoc: 2400).
  legs addLast: (connectionPoint createLeg: #Simplex direction: #Downstream to: '5678' qoc: 2400).
  connectionPoint freeCp.
  self dpsTrace: (Array with: connectionPoint with: legs).

Netw2MngrServer1>>createCp:with:for: (MngrServ<MngrClient)
createCp: typ with: respLev for: aClient
  " Create a ConnectionPoint. The legs must all be of the same type. "
  " type {#Simplex, #Duplex}. "
  " respLev is one of {#OnlyNack, #AckNack}. "
  | cp |
  cp := Netw2ConnPoint1 new.
  cp openCPFor: aClient type: typ responseLevel: respLev.
  ^cp

Netw2MngrServer1>>freeCp: (MngrServ<MngrClient)
freeCp: aCp
  " This message is used by the SockClient (service logic) to release "
  " a ConnectionPoint with all its legs and all its associated resources. "
  aCp freeCp

Netw2MngrServer1>>openSocketFor:responseLevel: (MngrServ<MngrClient)
openSocketFor: client responseLevel: lev
  " Requests the initialization of a Socket. level is one of {#OnlyNack "
  " #AckNack} This message must be the first message received by the Socket "
  " object, and cannot be received at any other time. "
  self dpsTrace: 'Message Ignored'.
\textbf{Netw2MngrServer1>>allocate:load: (AP<A)}

\textbf{allocate: type load: load}

\begin{quote}
"This message allows the Allocator to allocate resources in the AccessPoint"
"for a new call with a certain load (anInteger) of a given type (aSymbol)."
"Return true iff load can be accepted, otherwise false."
\end{quote}

^Previous == true

\textbf{Netw2MngrServer1>>deallocate:load: (AP<A)}

deallocate: type load: load

\begin{quote}
"This message allows the Allocator to release resources in the AccessPoint"
"for a certain load (anInteger) of a given type (aSymbol). Return true"
"if load can be accepted, otherwise false."
\end{quote}

self dpsTrace: (Array with: type with: load).

\textbf{Netw2MngrServer1>>loadCheck:for: (AP<ALM)}

loadCheck: type for: load

\begin{quote}
"This message can be sent to an Access Point to evaluate if it can"
"accept a new call with a certain load (anInteger) of a given type"
"(aSymbol). Return true iff load can be accepted, otherwise false."
"Note that this message does not allocate any resources in the Access"
"Point, it just checks if the Access Point could accept the load."
\end{quote}

Previous := Previous ~~ true.

^Previous
TABLE OF CONTENTS

Chapter 1  Introduction........................................................................................................ 2

Chapter 2  Overview of the Network Module................................................................. 4

Chapter 3  Network export models summary................................................................. 8
  3.1  Network Connection Model (Netw2ConnEXP) {Export Model}.............. 8
    3.1.1  Area of Concern ............................................................................................. 8
    3.1.2  The Roles ......................................................................................................... 9
    3.1.3  Interaction Scenarios .................................................................................... 10
  3.2  Network Access Manager EXP (Netw2MngrEXP) {Export Model}........ 10
    3.2.1  Area of Concern ............................................................................................. 10
    3.2.2  The Roles ......................................................................................................... 11

Chapter 4  Detailed report of the monitored execution.................................................. 12
  4.1  Netw2MngrClient1 Use Case 1 {Role Model} ...................................................... 12
    4.1.1  Area of Concern ............................................................................................. 12
    4.1.2  The Roles ......................................................................................................... 13
    4.1.3  Interaction Scenarios .................................................................................... 14
    4.1.4  Message Sets .................................................................................................. 14

Chapter 5  Role Models of the Network Module......................................................... 16
  5.1  Network Connection Model (Netw2ConnRM) {Role Model}......................... 16
    5.1.1  Area of Concern ............................................................................................. 16
    5.1.2  The Roles ......................................................................................................... 17
    5.1.3  Interaction Scenarios .................................................................................... 18
    5.1.4  Message Sets .................................................................................................. 18
  5.2  Network Access Load Checking (Netw2ChkRM) {Role Model}..................... 20
    5.2.1  Area of Concern ............................................................................................. 20
    5.2.2  The Roles ......................................................................................................... 21
    5.2.3  Message Sets .................................................................................................. 21
  5.3  Network Access Point Allocation (Netw2APLRM) {Role Model}.................... 21
    5.3.1  Area of Concern ............................................................................................. 22
    5.3.2  The Roles ......................................................................................................... 22
    5.3.3  Message Sets .................................................................................................. 23
  5.4  Network Access Manager (Netw2MngrRM) {Role Model}.............................. 23
    5.4.1  Area of Concern ............................................................................................. 23
    5.4.2  The Roles ......................................................................................................... 24
    5.4.3  Message Sets .................................................................................................. 25
<table>
<thead>
<tr>
<th>Chapter 6</th>
<th>Object type specifications in the Network Module</th>
<th>26</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>Network Connection Model (Netw2ConnSPC) {Type Specification}</td>
<td>26</td>
</tr>
<tr>
<td>6.1.1</td>
<td>Area of Concern</td>
<td>26</td>
</tr>
<tr>
<td>6.1.2</td>
<td>The Roles</td>
<td>27</td>
</tr>
<tr>
<td>6.2</td>
<td>Network Access Manager SPC (Netw2MngrSPC) {Type Specification}</td>
<td>28</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Area of Concern</td>
<td>28</td>
</tr>
<tr>
<td>6.2.2</td>
<td>The Roles</td>
<td>29</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 7</th>
<th>Network export models</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1</td>
<td>Network Connection Model (Netw2ConnEXP) {Export Model}</td>
<td>31</td>
</tr>
<tr>
<td>7.1.1</td>
<td>Area of Concern</td>
<td>31</td>
</tr>
<tr>
<td>7.1.2</td>
<td>The Roles</td>
<td>32</td>
</tr>
<tr>
<td>7.1.3</td>
<td>Interaction Scenarios</td>
<td>33</td>
</tr>
<tr>
<td>7.1.4</td>
<td>Message Sets</td>
<td>33</td>
</tr>
<tr>
<td>7.2</td>
<td>Network Access Manager EXP (Netw2MngrEXP) {Export Model}</td>
<td>35</td>
</tr>
<tr>
<td>7.2.1</td>
<td>Area of Concern</td>
<td>35</td>
</tr>
<tr>
<td>7.2.2</td>
<td>The Roles</td>
<td>36</td>
</tr>
<tr>
<td>7.2.3</td>
<td>Message Sets</td>
<td>36</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Appendix 1</th>
<th>Implementation specifications for monitored execution</th>
<th>38</th>
</tr>
</thead>
<tbody>
<tr>
<td>App 1.1</td>
<td>Network Connection Model (Netw2ConnSPC) {Type Specification}</td>
<td>38</td>
</tr>
<tr>
<td>App 1.1.1</td>
<td>Network Connection Model {Implementation}</td>
<td>38</td>
</tr>
<tr>
<td>App 1.2</td>
<td>Network Access Manager SPC (Netw2MngrSPC) {Type Specification}</td>
<td>41</td>
</tr>
<tr>
<td>App 1.2.1</td>
<td>Network Access Manager SPC {Implementation}</td>
<td>42</td>
</tr>
</tbody>
</table>
INDEX

Netw2ConnLeg1 >> subclass: instanceVariableNames: classVariableNames: poolDictionaries: category: (compilation) .............................................................................................................................. 39

Netw2ConnLeg1 >> connPt: client: direction: to: qoc: (Leg<ConnPoint) ............................................. 40

Netw2ConnLeg1 >> freeLeg (Leg<ConnClient) ...................................................................................... 40

Netw2ConnLeg1 >> modifyDirection: (Leg<ConnClient) ..................................................................... 40

Netw2ConnLeg1 >> modifyQoc: (Leg<ConnClient) ............................................................................. 40

Netw2ConnPoint1 >> subclass: instanceVariableNames: classVariableNames: poolDictionaries: category: (compilation) .............................................................................................................................. 40

Netw2ConnPoint1 >> createLeg: direction: to: qoc: (ConnPoint<ConnClient) ................................. 40

Netw2ConnPoint1 >> freeCp (ConnPoint<ConnClient) .......................................................................... 41

Netw2ConnPoint1 >> freeLeg: (ConnPoint<ConnClient) .................................................................... 41

Netw2ConnPoint1 >> legFreed: (ConnPoint<Leg) ................................................................................ 41

Netw2ConnPoint1 >> openCPFor: type: responseLevel: .................................................................. 41

Netw2MngrClient1 class >> example1 (examples) ............................................................................. 43

Netw2MngrClient1 >> example1 (private) ......................................................................................... 43

Netw2MngrServer1 class >> default (instance creation) ................................................................. 43

Netw2MngrServer1 class >> initialize (class initialization) .......................................................... 43

Netw2MngrServer1 class >> subclass: instanceVariableNames: classVariableNames: poolDictionaries: category: (compilation) .............................................................................................................................. 43

Netw2MngrServer1 >> allocate: load: (AP<A) .................................................................................. 44

Netw2MngrServer1 >> createCp: with: for: (MngrServ<MngrClient) .................................................. 43

Netw2MngrServer1 >> deallocate: load: (AP<A) ................................................................................ 44

Netw2MngrServer1 >> freeCp: (MngrServ<MngrClient) ................................................................. 43

Netw2MngrServer1 >> loadCheck: for: (AP<ALM) ............................................................................ 44

Netw2MngrServer1 >> openSocketFor: responseLevel: ................................................................. 43

(MngrServ<MngrClient)