Author: Vesa-Matti Puro / Timo Pärnänen

Document: Sw module documentation

Date: 13/01/1999

Version: 0.6

SW

The sw module provides the framework for ATM switch software. It provides facilities to access the cards, physical interfaces, logical signaling interfaces and other resources in the switch.

1 Introduction

The sw module implements a main program for ATM switch software. It provides access to switch resources by delegating operations to proper modules. Administrators have runtime access to configure switch parameters and resources via CORBA interface. Standardized management protocols are used to get information of switch parameters.

2 Architecture

The figure 1 shows the switch architecture. Management protocol implementations, SNMP (Simple Network Management Protocol [1]) and ILMI (Interim Local Management Interface [2], [3]), and configuring interface implementation are CORBA adapters to support access to ATM MIB and other system variables (ILMI is used only in UNI links). Routing resources are implemented in other module and it is distributed also with CORBA. The switch class has different prototypes and common objects as attributes, e.g. SS7 (Signaling System No. 7) stack, and cloneable prototypes of signaling protocols.

The architecture of the switch framework consists of following classes or collections of classes: system, switch, routing, card, physical interface, logical signaling interface, subscription, call, virtual path and virtual channel. In the following, each of these entities are described further (some classes are just ideas without concrete implementation)

- System provides a singleton for a global access point for the switch framework. The system can contain several switches that can be accessed through the system. The System may be implemented using several separate singletons if needed.

- Switch is an object that provides an access point to a switch and resources of the switch. The interface of the switch provides access to line interface cards, physical interfaces in those cards, logical signaling interfaces in those physical interfaces and

virtual paths and channels in physical interfaces in addition to other physical or logical resources.

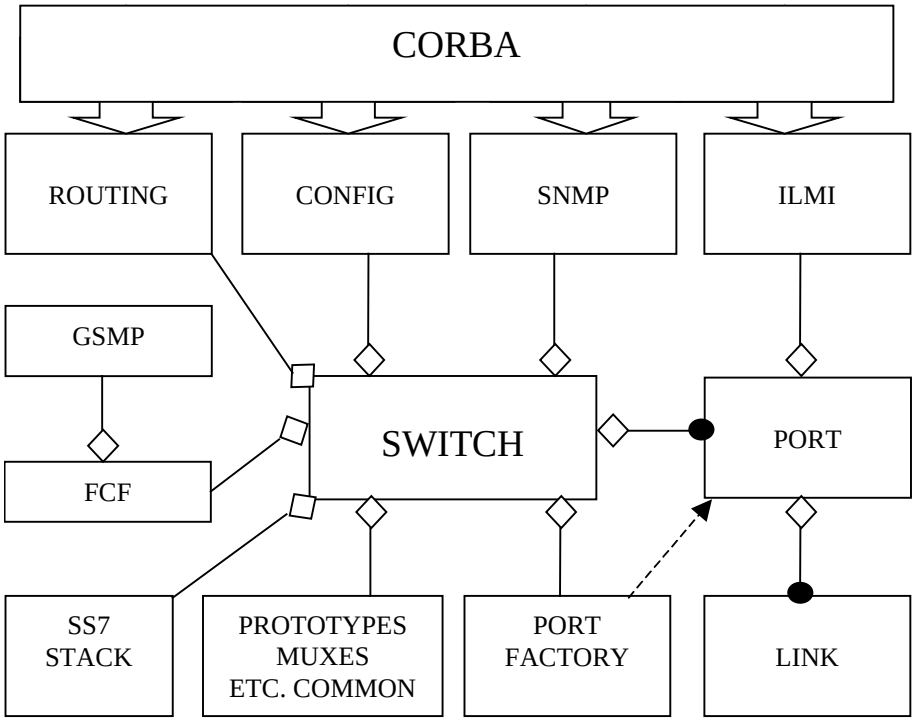

Figure 1. Switch architecture.

- Routing is an object that makes routing decisions based on the information given from the switch. The routing contains a static route database. The switch informs routing about dynamic changes in routes such as congestion, blocking and route failures.

- Card is an object that models a physical line interface card attached to the switch. One card can contain one or several physical interfaces. Activation or deactivation of a card activates or deactivates physical interfaces contained in the card. Other operations performed on the card can be propagated to physical interfaces as well.

- Physical interface (port) is an object that models a physical network interface that is connected to the switching fabric. The physical interface has several properties that describe characteristics of the interface such as line speed and the number of virtual paths and channels supported.

- Logical signaling interface (link) is an object that models the signaling capabilities of the whole physical interface or one logical channel inside the physical interface. The logical signaling channel supports one of various signaling protocol stacks such as UNI or NNI signaling stack. The UNI signaling stack may contain ATM, CPCS, UNI-SSCF and then one of access signaling protocols such as UNI 3.1, UNI 4.0 or

Q.2931. The NNI signaling stack may contain ATM, CPCS, NNI-SSCF and network signaling protocol (BISUP).

- Subscription is an object that models subscribed services in local interfaces. It contains a user service profile.

- Call is an object that models the ongoing call and resources associated to it. The call can be SVC or PVC.

- Virtual path is an object that models virtual path connection in a physical interface. This object is used for unallocated and allocated virtual paths. Virtual paths can be switched if that is supported by the hardware.

- Virtual channel is an object that models virtual channel connection in a physical interface. This object is used for unallocated and allocated virtual channels. Virtual channels can be switched if that is supported by the hardware.

A port factory creates ports using registered port prototypes (identified with string). A fabric control function (FCF) is a generic interface, which hides the concrete switching protocol or API, e.g. ATM FCF implementation uses GSMP (Generic Switch Management Protocol [4]) to make fabric connections in a switching hardware.

3 Implementation details

There are three major collections of classes implemented in the sw module. Those are port, link and fcf (fabric control function). The figure 2 describes a class hierarchy of port and link classes. The fcf hierarchy is presented in the figure 3.

3.1 Port and link

A base class of the port hierarchy stores link objects, mapped with link number as a key. The link class hierarchy encapsulates only an actual signaling protocol, which is implemented using the signaling framework (see TOVE signaling framework document). Inherited link classes are used to instantiate and set the selected protocol. Lower layer protocols must be encapsulated in inherited port class. For example the swAtmPort subclass stores SAAL (Signaling ATM Adaptation Layer) link objects, which encapsulates SAAL sub-layer protocols.

The ATM port implements also the swPortConfig interface, which is used by GSMP to configure port parameters and VPI/VCI values. The ATM port has the ILMI CORBA adapter as an attribute. The purpose of this adapter is to provide an access to ATM MIB for distributed ILMI agents. There is no any concrete MIB object in sw, but ILMI adapter delegates requests to the object where information exists.

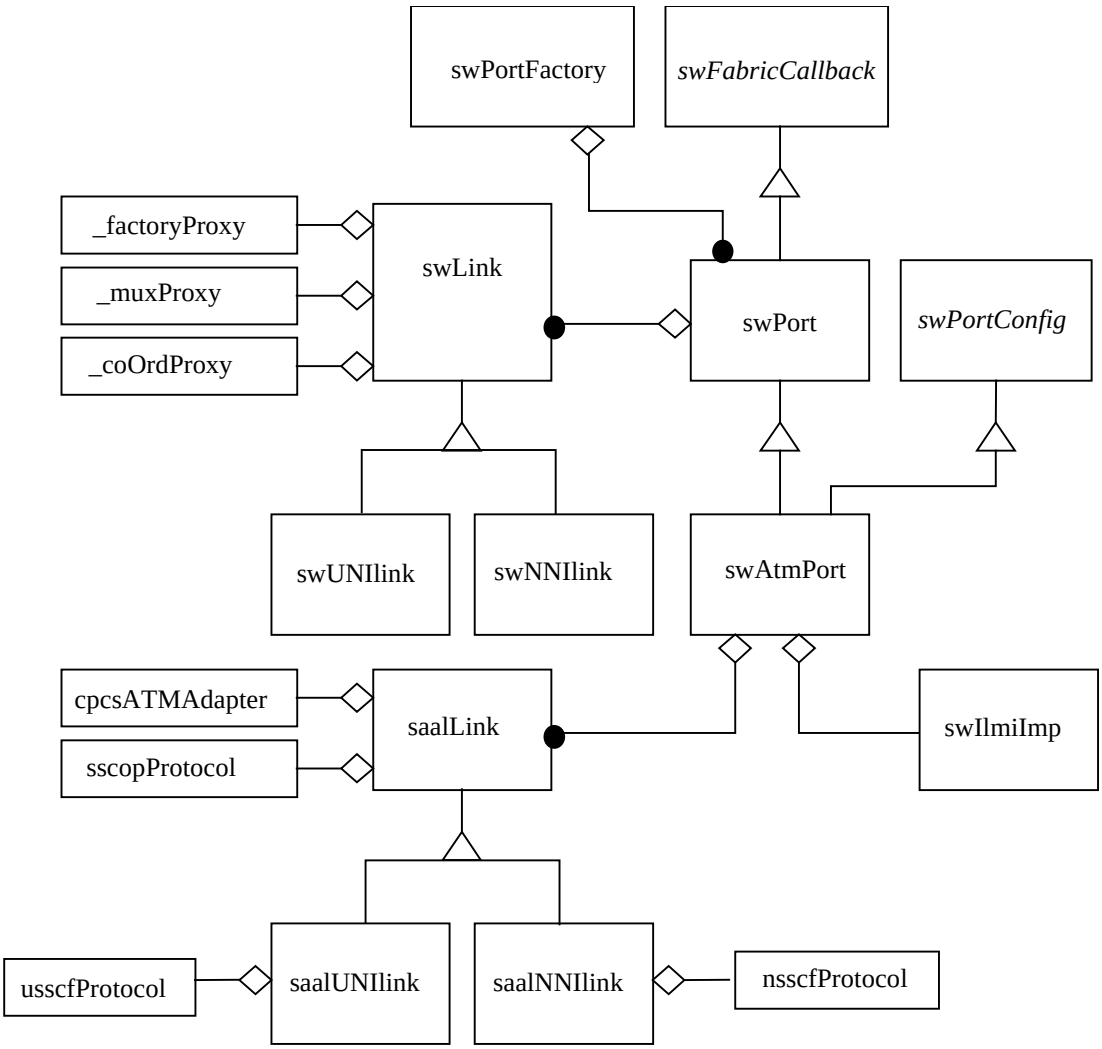

Figure 2. Port and link.

3.2 FCF

The Fabric Control Functions (FCF) is used to hide switching functions from the Call Control and call procedures at all. The FCF is not based any recommendations or specifications, it is only a wrapper between the call procedure and switching fabric. When ATM links are going to be connected, FCF interprets connect functions from the call procedure and uses GSMP to control the switch to make a virtual connection in the switch. In case of narrowband links, some other switching methods (e.g. FSR API) are used in FCF.

The FCF makes connections based on attributes in information elements. All information elements which are used in the connect operation are derived from the ieConnectInfo base class. The figure 3 shows an information element hierarchy for the connect operation. The switch has a connect method with a base class type of objects as parameter, and double dispatching technique is used to type check for FCF (see chapter 3.3).

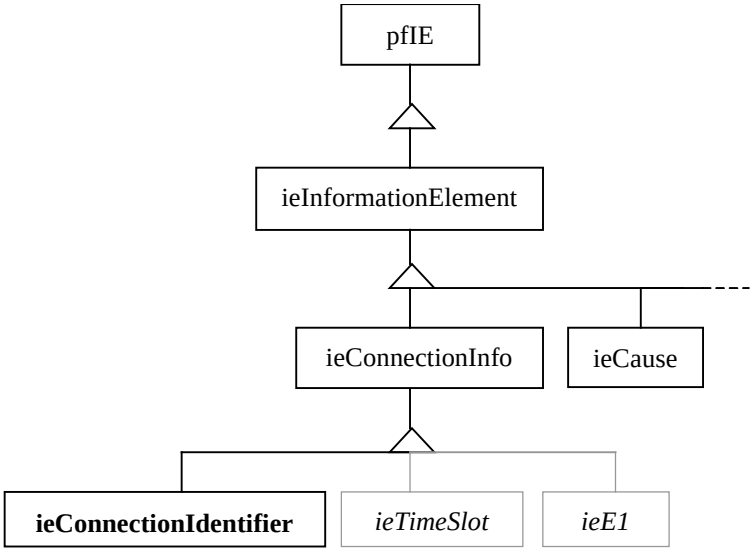

Figure 3. Information elements for connect operation.

The figure 4 shows a class hierarchy for ATM FCF. The swFCF base class implements three generic interfaces. The swConnectIface implements different connect methods, swFabricCallback is used to get callbacks (success or failure) to connect operations, and swEventIf is used to get generic state information of the fabric and its ports.

The swATMFCF class implements also the swConfigControl interface which is used to

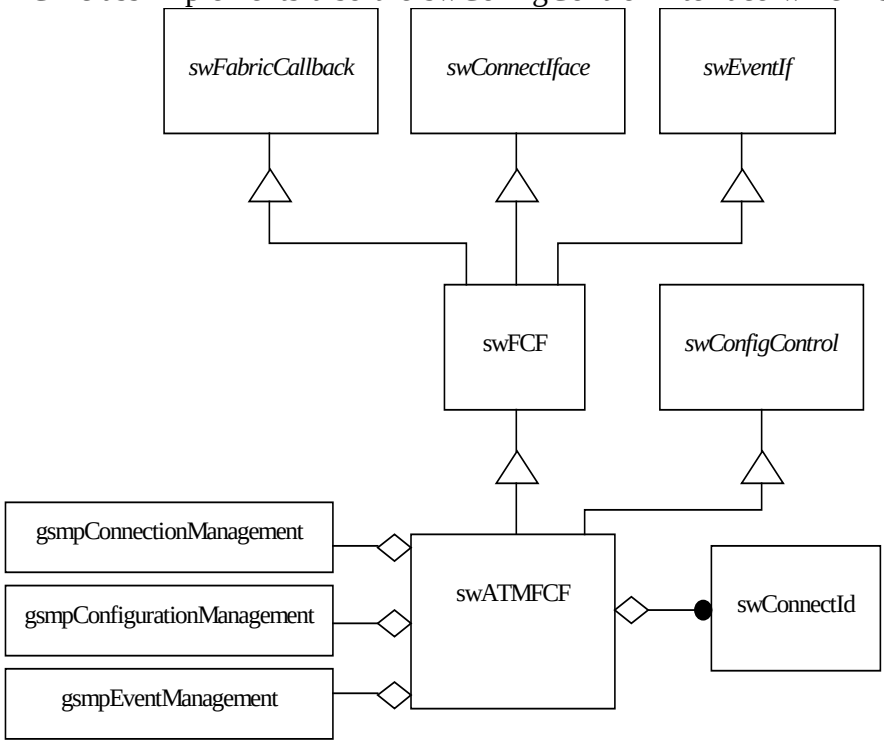

Figure 4. ATM FCF.

set some switch attributes and to provide configure objects for each port. The ATM FCF

has the GSMP connection, configuration and event management object as attributes to handle corresponding actions in the switching fabric using the GSMP protocol. The swConnectId class is used to store connection information and handle responses (success or failure) to switch command from GSMP.

3.3 Switching operation

The following figure shows connect (point-to-point) operation steps starting in a signaling interface between the UNI protocol and call control.

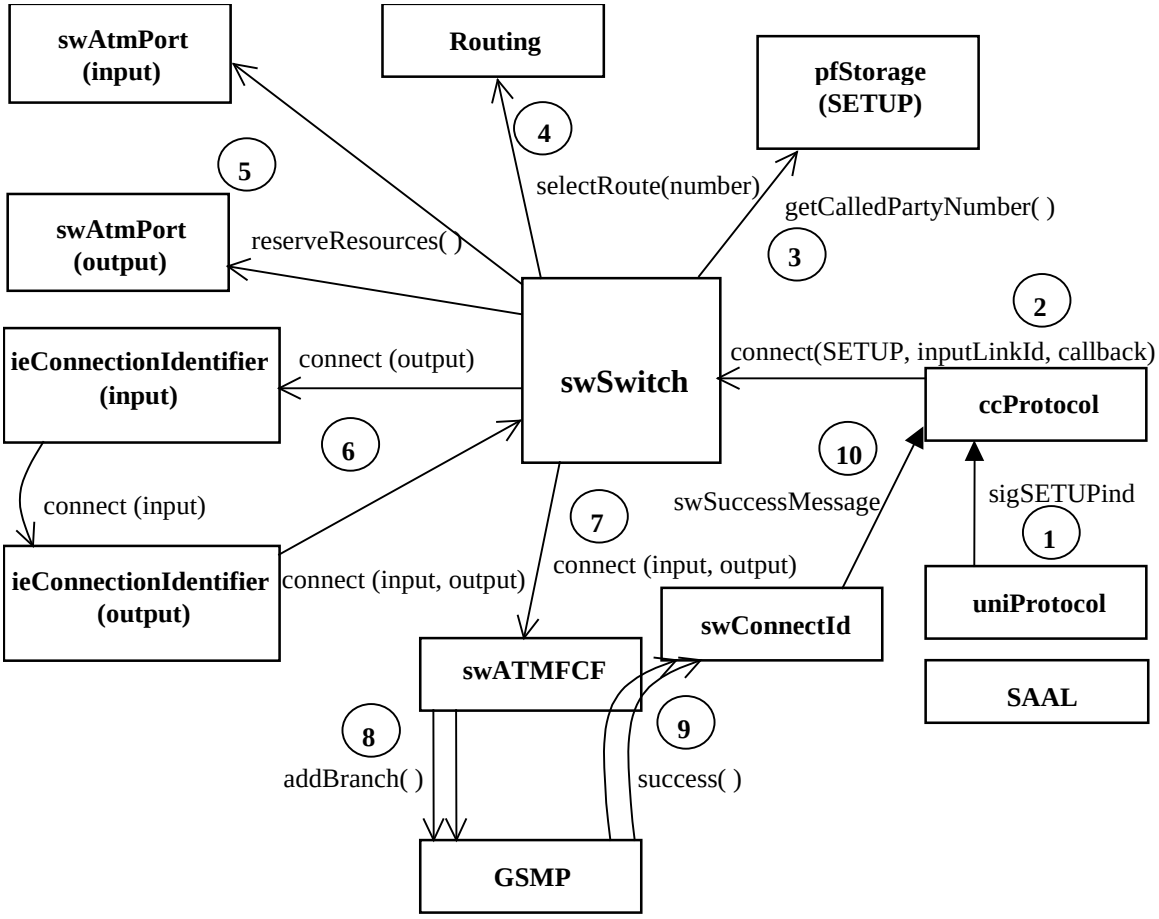

Figure 5. Point to point connect using UNI ATM signalling.

Steps of the point-to-point connect operation in UNI ATM signaling:

1. UNI protocol sends sigSETUP indication primitive to the call control. The primitive includes decoded parameters from the UNI SETUP pdu.

2. Call control calls a connect method from the switch with the SETUP storage, input link identifier and proxy for the call control (callback object) as parameters.

3. Switch object gets a called party number from the SETUP object.

4. Routing module is used to find a destination port/link. The route selection based on called party number.

5. Resources for input and output connections are reserved from corresponding port objects. The resource information (e.g. VPI/VCI values) is stored up information elements (input/output).

6. Because the switch object has a base class type of information elements for input and output connections, double dispatching is used to recognize the actual type of information elements (in ATM connection the type is ieConnectionIdentifier).

7. When the connection type is clarified the proper FCF (here ATM FCF) is called to make the connection.

8. ATM FCF uses the GSMP protocol to add unidirectional branches to the switch. Due to unidirectional switch command, two branches must be added per point-to-point connection.

9. SwConnectId handles responses from GSMP.

10. When responses for both add branch commands are received, the switch sends a callback message to the call control to indicate that fabric connection is successfully done.

3.4 Interfaces

Following interfaces list ideas of methods that could be necessary in corresponding objects in the sw module. These requirements are founded in different standards and recommendations. Not all interfaces are implemented and some are partially implemented.

The switch has the following interface:

```
class swSwitch
{
// System Group
// based on UNI31 4.5.1
        string getDescription(void) const;
        string getObjectIdentifier(void) const;
        string getUpTime(void) const;
        string getContact(void) const;
        string getName(void) const;
        string getLocation(void) const;
        int getServices(void) const;
```

```
// system variables
        int getPointCode(void) const;
        string getPrefix(void) const;
        int getESI(void) const;

// based on GSMP 2.0 version 7.1
// maybe these can be implemented in the swVersion class that could be inherited to
// swSwitch
        int getFirmwareVersion(void) const;         // 16 bit number
        int getSwitchType(void) const;              // 16 bit number
        int getSwitchName(void) const;              // 48 bit number

// based on System Group sysDescr
        string getHWname(void);
        int getHWversion(void);
        string getOSname(void);
        int getOSversion(void);
        string getNWname(void);
        int getNWversion(void);

// port management
        // Port types FSR155 / FSRE1 / FSRTAXI etc.
        void addPort(int portNumber_, const string &portType_);
        void removePort(int portNumber_);
        swPort *findPort(int portNumber_) const;
};
```

The port has the following interface:
(function is equal with UPD connection)

```
class swPort
{
// link management
        void addUNIlink(int linkNumber_);  // maybe link number can be same as vpi
        void addNNIlink(int linkNumber_, int destinationPointCode_);
        void removeLink(int linkNumber_);
        swLink *findLink(int linkNumber_) const;

// Link number and port number are combined together. This link identifier is used also in
// a crossConnMux with port number
// reservation of paths
        swPath *reserveVPI(void);
        swPath *findVPI(int vpi_);
        void releaseVPI(swPath *vpi_);
```

```
// port management (affects to port status)
        void bringUp(void);                 // bring up once or keep it up
        void takeDown(void);                // take down and keep it down
        void resetInputPort(void);          // connections are removed

// loopback control (affects to port status)
        void setInternalLoopback(void);     // output cells are relayed to input
        void setExternalLoopback(void);     // input cells are relayed to output
        void setBothwayLoopback(void);      // both of the above

// event management (affects to line status)
        void portUp(void);                  // event happened in the port
        void portDown(void);                // event happened in the port

// Physical Interface UNI MIB Attributes
// based on UNI31 4.4.2
        int getTransmissionType(void);
        int getMediaType(void);
        int getOperationalStatus(void);

// ATM Layer Interface UNI MIB Attributes
// based on UNI31 4.4.3
        int getMaxVPCs(void);
        int getMaxVCCs(void);
        int getConfiguredVPCs(void);
        int getConfiguredVCCs(void);
        int getMaxVPIbits(void);
        int getMaxVCIbits(void);
        int getType(void);
        int getVersion(void);

// GSMP 2.0 version 7.2
        int getMinVPI(void);
        int getMaxVPI(void);
        int getMinVCI(void);
        int getMaxVCI(void);
        int getReceiveCellRate(void);
        int getTransmitCellRate(void);
        int getPortStatus(void); // available, unavailable, internal/external/bothway lb
        int getPortType(void);
        int getLineStatus(void);            // up, down or test
        int getPriorities(void);
        int getPhysicalSlotNumber(void);
        int getPhysicalPortNumber(void);

        pfBoolean isVPswitchingSupported(void);
```

```
        pfBoolean isMulticastLabelsSupported(void);
        pfBoolean isLogicalMulticastSupported(void);
        pfBoolean isLabelRangeSupported(void);
        pfBoolean isQoSsupported(void);

// ATM Layer Statistics
// based on UNI31 4.4.4
// this has to be implemented using callback
        int getReceivedCells(void);
        int getDroppedReceivedCells(void);
        int getTransmittedCells(void);
};
```

The link has the following interface:

```
class swLink
{
// requirements from BISUP and routing
        int getPortNumber(void);
        int getPeerPointCode(void);
        string getPeerPrefix(void);

// ATM Layer Interface UNI MIB Attributes
// based on UNI31 4.4.3
        int getMaxVPCs(void);                   // check BISUP IEs for these parameters
        int getMaxVCCs(void);
        int getConfiguredVPCs(void);
        int getConfiguredVCCs(void);
        int getMaxVPIbits(void);
        int getMaxVCIbits(void);
        int getType(void);
        int getVersion(void);
};
```

The path has the following interface:

```
class swPath
{
// reservation of VCI
        swChannel *reserveVCI(void);
        void releaseVCI(swChannel *vci_);

// Virtual Path UNI MIB Attrbutes
// based on UNI31 4.4.5
        int getVPI(void);
        int getOperationalStatus(void);
```

```
        int getTransmitTrafficDescriptor(void);      // check IEs for these parameters
        int getReceiveTrafficDescriptor(void);
        int getTransmitQoSclass(void);
        int getReceiveQoSclass(void);
};
```

The channel has the following interface:

```
class swChannel
{
// Virtual Channel UNI MIB Attributes
// based on UNI31 4.4.6
        int getVPI(void);
        int getVCI(void);
        int OperationalStatus(void);
        int getTransmitTrafficDescriptor(void);
        int getReceiveTrafficDescriptor(void);
        int TransmitQoSclass(void);
        int ReceiveQoSclass(void);
};
```

4 Features implemented

This release of the sw module implements ATM versions of port and FCF objects. UNI and BISUP signaling protocols are supported in UNI/NNI link classes. Connect methods for switching operation support only ieConnectionIdentifier type of information elements used in ATM connection.

Not all classes, mentioned in a previous chapter, are implemented. The path and channel classes are replaces with integer values. The switch, port and link classes are implemented, but they do not support all methods listed in classes in the previous chapter.

Also the system, card, call and subscription (and virtual path and channel) classes are not implemented in this release of the sw module.

5 Limitations

Error handling is insufficient in several methods in classes of the sw module. In general freeing of resources is incompletely implemented. Detailed code reviews and testing phase are needed to figure out possible problems.

6 Future development

Narrowband objects like the port, link and FCF will be implemented (see an example in Appendix chapter). The narrowband connect operation requires also new information elements (e.g. ieTimeSlot, ieE1) and corresponding FCF may use some other protocol or API than GSMP to control the switch. Maybe some ideas mentioned in previous chapters get concrete implementations in future.

7 References

[1] Case, J., Fedor, M., Schoffstall, M., Davin, J., "A Simple Network Management Protocol (SNMP)", RFC 1157, May 1990.

[2] The ATM Forum Technical Committee, "Integrated Local Management Interface (ILMI) Specification Version 4.0, af-ilmi-0065.000, September 1996.

[3] The ATM Forum Technical Committee, "ATM User-Network Interface Specification (v3.1), Section 4: Interim Local Management Interface Specification".

[4] Newman, P., Edwards, W., Hinden, R., Hoffman, E., Ching, F., Lyon, T., Minshall, G., "Ipsilon's General Switch Management Protocol Specification Version 2.0", RFC 2297, March 1998.

8 Appendix

Following list shows steps how to implement new port/link/FCF combination. ISDN (Integrated Services Digital Network) is used as an example.

1. Create a new signaling link class (swISDNlink) by inheriting it from swLink class. Implement methods for prototype handling and setConduits to instantiate signaling protocol.

2. Create a new port class (swNarrowBandPort) by inheriting it from swPort class. Store corresponding link (lower layer protocols like LAPB) classes in some collection structure in new port class. Implement some addLink method where both link objects (for lower layer and upper layer) are created, stored and connected together. Remember to implement a clone method.

3. Register new port type to the port factory in swSwitch.

4. Store prototypes of new link classes and make clone methods (named create) in swSwitch.

5. Create a new information element (e.g. ieTimeSlot or ieE1) by inheriting it from ieConnectionInfo class in ie module. Add corresponding connect methods for double dispatching.

6. Add new connect/disconnect methods, which has new information elements (mentioned in previous step) as parameters, to swConnectIface

7. Create a new FCF object (swNarrowBandFCF) by inheriting it from swFCF class. Store an instance of suitable switching protocol or API in this new FCF. Implement needed interfaces from swif module (default implementation is empty code in swFCF).

8. Store an instance of new FCF in swSwitch.

9. Implement new connect methods (mentioned in step 6) to swSwitch. These methods delegates connect method calls to the right FCF instance.