GUI For Managing the Equipment In An ATM Switch System

Mikael Wallin

Contents

1.0 Introduction

This master thesis was performed at Ericsson Telecom AB, a unit of Ericsson that develop ATM switches. The purpose of the thesis was to design and implement a Graphical User Interface (GUI) for managing the equipment (i.e. subracks and boards) in a ATM switch. Included in the task was to implement a server application to access the Management Information Base (MIB) for the equipment of the ATM switch.

The equipment management area is concerned with the control of the physical hardware rather than any of the traffic that goes through the system. The equipment management functions are used to control the hardware and those parts of the system that are directly related to it.

Today there exists software to locally control the switch via a pure textual WWW interface. An operator can use a portable computer that supports a standard web browser to perform equipment management. The portable computer is connected to the ethernet connector of the ATM switch, and can thereby control the switch through http-based communication.

The web browser on the computer communicates with a "local management server" in the switch which will answer to management requests. The data in the responses is either html/images-files stored in the file system or html-code generated within the ATM switch. These html pages are created using scripts designed in erlang and contains the managed data. All the information that is displayed to the operator is fetched from the management information base (MIB) in the ATM switch.

An operator using WWW interface has to step through many pages to perform all the different actions on the hardware. It is not even possible to view the equipment as graphical objects, which is the main motivation for this thesis. Implementing a GUI will also reduce the large amount of different menus in the user interface.

The idea for the user interface is to use the facility to create custom GUI components with Java and use some layout manager to put them on the screen. Each GUI component describes one physical hardware unit.

This report covers the following studies:

- Erlang programming language. The server side should be implemented in Erlang for real-time requirements.
- Open Telecom Platform (OTP), the development environment at Ericsson Telecom.
- Java programming language for object oriented design of the GUI.
- ATM switch hardware architecture.
- ATM switch software architecture.

The remainder of the report is structured as follows:

Chapter 2 of the report is a description of the hardware in the switch. It also describes the meaning of the tasks that an operator can perform on the hardware.

Chapter 3 describes the tables in the MIB used for handling the hardware and the links on the PIU's.

Further, the present interface to control the switch hardware is described in chapter 4. A detailed list of the requirements of this project is also given in this chapter.

Chapter 5 presents the client-server architecture of the application. The functionality of the Erlang modules that is implemented to interact with the management resources is also described. The Java components that is implemented for the GUI is presented. At the end of the chapter the protocol used to send messages between the Java client and the Erlang server is described.

The implementation of the Erlang server and the Java client is described in chapter 6. This chapter also describe how to operate the interface.

Chapter 7 concludes the report.

2.0 Switch Hardware

AXD 301 switching system hardware consists of these main components [4]:

- Cabinets for housing equipment subracks and external power equipment
- Subracks for housing (circuit) boards
- Boards for realising the switch functionality.

2.1 Cabinet

AXD 301 switching system hardware is installed in a switching cabinet, as shown in Fig.1. The cabinet contains a Power Distribution Unit (PDU) for regulating the power supply to the boards installed in the equipment subracks.

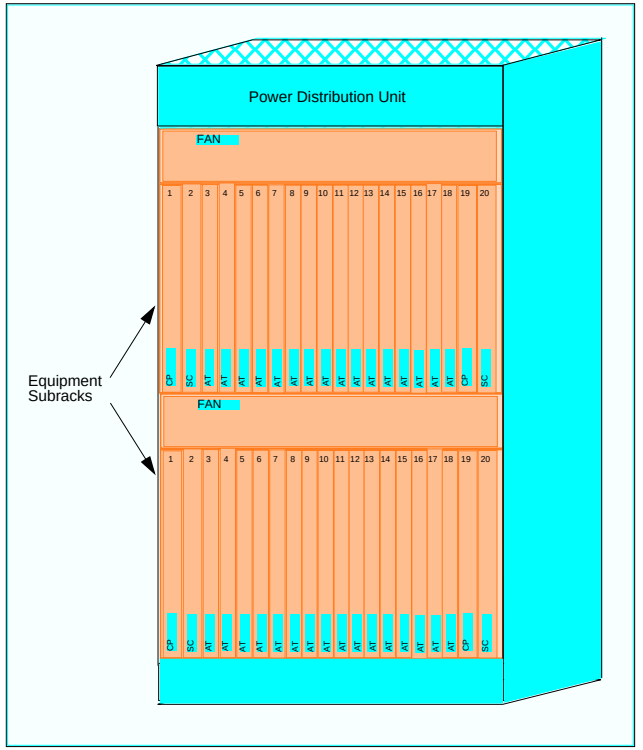

Fig.1. AXD 301 Switching System Cabinet (Front view).

2.2 Subrack

A switch is made up of various boards stored in a subrack. Fig.2. shows the front view of a fully equipped 10 Gbit/s switch. The boards labelled AT in this figure are the Asynchronous Transfer Mode (ATM) Termination Boards.

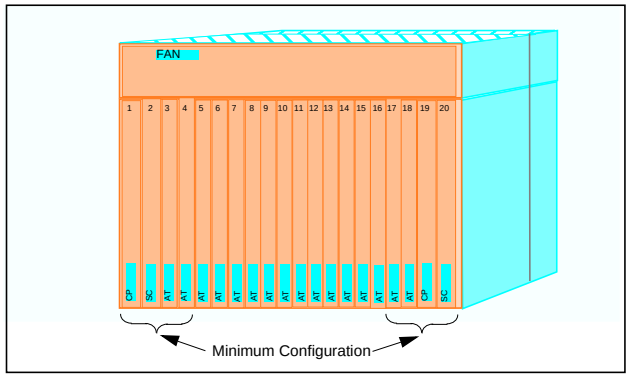

Fig.2. 10 Gbit/s switch (front view)

There is a backplane in the switch (shown as the dotted line down the side of Fig.2.), which means that there is one set of boards accssible from the front of the switch, and a different set of boards accessible through the back (see Fig.3.).

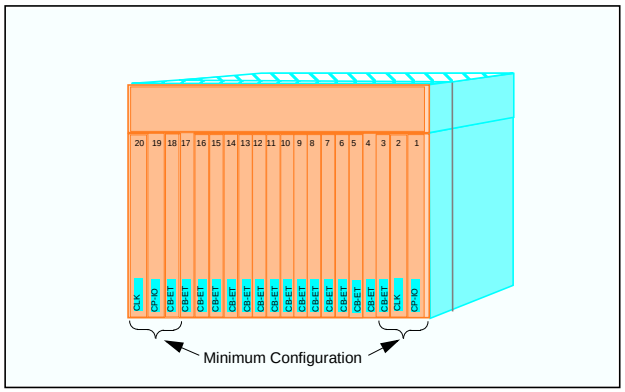

Fig.3. 10 Gbit/s Switch (back view).

2.3 Plug-In Units

The individual boards in the system are known as Plug-in Units (PIUs). PIU is the level you deal with when you are repairing or testing the system.

It is possible to test the boards if they are currently blocked. This generates an event which shows the details of the test. The boards accessible from the back of the switch are called the Connection Boards (CBs). Fig.4.shows what this would look like if you could take a slice through the switch. At the front of the switch you have the larger of the two boards, and at the back you have the connection board and all of the cabling.

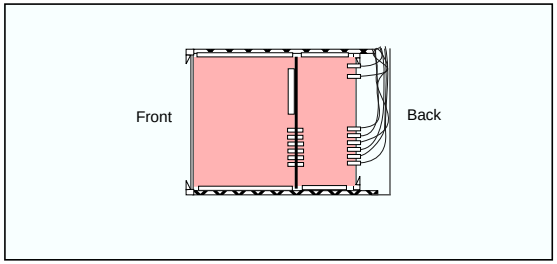

Fig.4. Side view of boards in the switch.

On the front of each board there are three Light Emitting Diodes (LEDs), as shown in Fig.6. Each LED is of a different colour to help you to distinguish the meanings. The meanings are constant for all the boards in the switch.

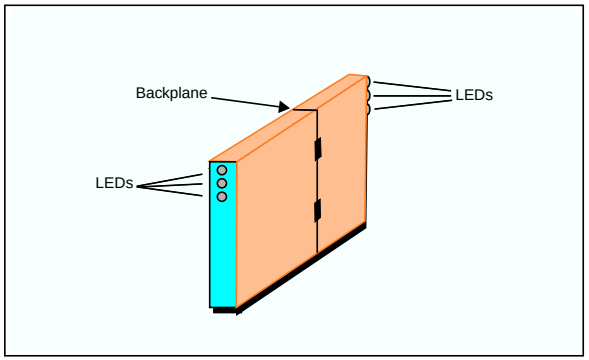

Fig.5. Position of LEDs on the boards.

The red LED lights up when there is an alarm on this board (the first LED from the top in the figure). The green LED lights up when the board's internal (5 volt) power source is working (the LED in at the bottom).

The most important LED is the yellow (in the middle) one. This one is lit when the board is blocked, indicating that the board can be removed. When this LED is OFF, the board has not been blocked and so should not be removed from the subrack.

2.4 Types of AXD 301 Boards

There are various types of boards that can be included in the Switch. These are described in TABLE 1. The switch must have Control Processors (CPs) and Switch Cores (SCs) boards. In addition there may be one ABR and two Clock (CLK) boards. The rest of the boards are AT (ATM Exchange Termination) boards of different types. Fig.2. and Fig.3. also shows the basic minimum configuration of boards that are required to realise the switching functionality of the AXD 301.

The boards are usually identified by labels which contain a barcode and the name of the board. These identities are shown in Fig.1., Fig.2., Fig.3. and the Identity column in TABLE 1.

Identity	Name	Description
SC	Switch Core (SC) board	The SC board provides functionality for the space switching of ATM cells. Because the switch core has a redundant plane, each subrack must be equipped with two SC boards.
CLK	Clock (CLK) board	The CLK board provides switch timing and synchronisation functionality. The board is equipped with four dedicated synchronisation inlets to support the network synchronisation functionality. For redundancy reasons, each subrack must be equipped with two CLK boards.
CP	Control Processor (CP) board	The CP board provides the switch with a Central Processing Unit (CPU). For redundancy reasons, each subrack must be equipped with two CP boards.
CP-IO	CP Input Output (IO) board	Provides and input/output logic between devices and the CPs CPU, and IO control between the CPs CPU and its harddisk. The board also provides customer and support equipment alarm inlets. For redundancy reasons, each subrack must be equipped with two CP-IO boards.
AT	ATM Termination board	The AT board terminates the switch redundancy, and manages ATM cell buffering.
CB_ET	Exchange Termination Connection Board (ET-CB)	The ET-CBs provide different line exchange termination functionalities depending on the physical ATM interfaces being supported, such as 34 Mbps PDH and 155 Mpbs SDH. Each ET-CB provides one or more ports for connecting physical links.
CB_ABR	Available Bit Rate server Connection Board (ABR-CB	The ABR-CB manages the ABR service in the switch. Only one ABR-CB may be installed in a subrack.

TABLE 1. Board Types in the AXD 301 Switching System

2.5 Equipment Modules

Although the individual physical PIUs are the basic level when you are testing or repairing the system, when you are performing any other task, the basic level is the Equipment Module (EM).

An EM is a logical construct - that is, it does not really exist as a physical entity. However, the EM allows you to group the following boards together, and treat the whole EM as a single object:

- An ET-CB and AT, that can be treated as a single ET EM.
- A CP-IO and CP, that can be treated as a single CP EM.
- Two CLK PIUs and two SC PIUs, that can be treated as a single SC EM.
- A ABR-CB and AT, that can be treated as a single ABR EM.

Being able to treat an EM as a single object means that whenever you use that EM for configuration, fault localization, etcetera, the functions will be carried out automatically on all the PIUs within the EM. This also means that all management operations other than the basic repair and testing of physical PIUs, are carried out on logical representations of PIUs within the EM. The only PIU identity that has any meaning for these management operations is the logical PIU identity belonging to an EM. Fig.6. shows you the different physical and logical views of the system.

The physical view shows how two PIUs are connected via the backplane, and that each PIU has its own set of Light Emitting Diodes (LEDs). Each pair of PIUs fits into one slot in the subrack. In the logical view, the following types of EM are shown:

- The Exchange Termination (ET) EM. Each slot in an ET EM is occupied by two PIUs. The AT board is placed in the front slot and the ET-CB is placed in the back slot.
- The Control Processor EMs are made up of a single slot with a CP-IO board and a CP board. The Control Processors are linked to the closest ET EMs.
- The Switch Core (SC) EM. This is made up of two slots per subrack. Each slot consists of a CLK board and an SC board.

- The ABR EM. This is made up of two slots per subrack. Each slot consists of a ABR-CB and an AT board.

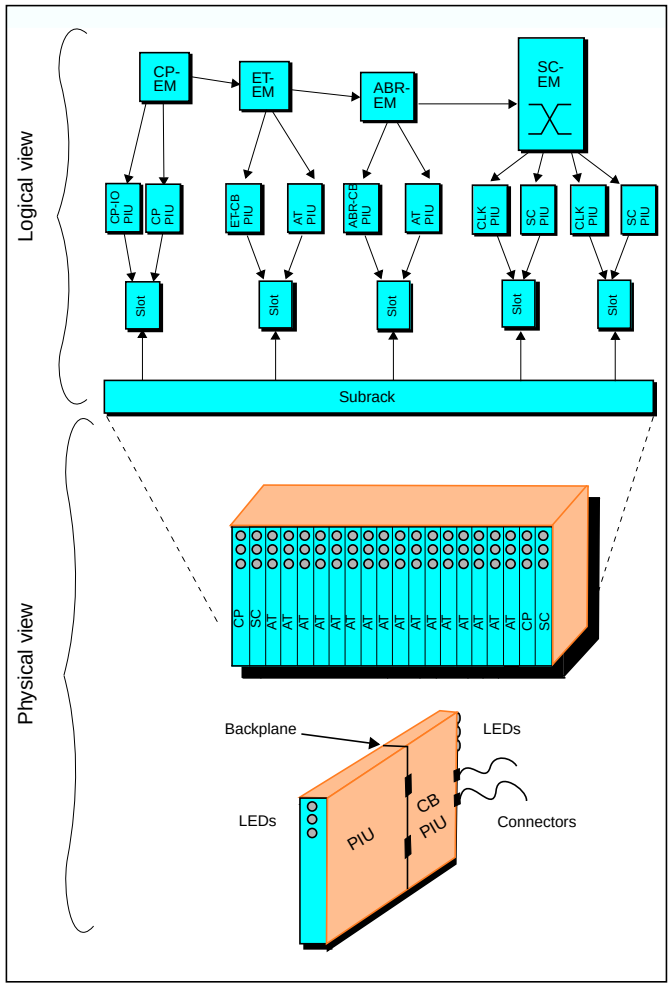

Fig.6. The different views of the system.

The ET-CBs provide different line exchange termination functionalities depending on the types of physical interfaces being supported. Each ET-CB provide one or more

ports for connecting physical links. You can create ET EMs based on different physical interface types, such as:

- 155 Mbps electrical
- 155 Mbps optical
- 34 Mbps
- 622 Mbps optical

When the Equipment Module is created, information about the EM is stored in the EM table (see Section 3.1.4). Because the EM is a logical (rather than a physical) construct, it only really exists as entries in tables.

3.0 EQM MIB

This chapter describes the Management Information Bases used for handling the equipment of the ATM switch.

Information about the equipment that makes up the system is held in various database lists. The management block contain several tables forming a Management Information Base (MIB).

There exists two Management Information Bases in the AXD 301 switch system, one for handling the PIUs and EMs (equipment management MIB) and one for handling the interfaces, i.e. the links on the ET-CBs. Information from the MIBs can be fetched through a set of instrumentation functions, an OTP-SNMP Agent interface.

3.1 Equipment Management MIB

It is through the interface to this MIB possible to create, block, deblock, delete and test the EMs and PIUs in the system. Some of the tables are described in [6] are listed below.

3.1.1 The subrack table

There are one or two rows in this table, one for each subrack in the system. The rows in this table are created as a result of create or increase SC EM. An entry of the table contains information a bout a subrack such as.

- subrackId. Identifier of a certain subrack.
- subrackUserLabel. A user friendly subrack name.
- subrackPos. The subrack position in the cabinet.

3.1.2 The slot table

This table contains one row for each slot in the system. There exists twenty slots for each row in the subrack table. A row in this table is identified by both the subrack Id and the slot number. An entry in this table contains information about a slot in the subrack:

- slotSubrackId. A reference to the associated subrack.
- slotNo. Identifies the slot position within the subrack.
- slotSide. The value of this object identifies the subrack side of the slot.
- slotState. This object can have four different values: usedOccupied (4) means that the slot is used by an Equipment module and equiped with HW, unusedOccupied (3) means that the slot is not used by any Equipment module but there is HW inserted, usedEmpty (2) means that the slot is used but there are no HW inserted and unusedEmpty (1) means that the slot is not used by any Equipment module and no HW is inserted.
- slotEm. This value together with the slotSubrackId identifies the Equipment module that uses this slot.

- slotExpPiutype. The type of Plug-In Unit expected to use this slot.

3.1.3 The SC EM table

This table contains only one row describing the Equipment module ScEm. It holds information about:

- ScEmType. The type of the switch core. When the Equipment module is created this value decides how many Plug-In Units that will be created and in which slots in the subracks they must be placed.
- scEmAdmState. The adminastrative state of the SC EM.
- scEmOpState. The operational state of the SC EM.
- scEmActivePlane. Indicates which plane is active, plane A or plane B.
- scEmActivePlaneMode. Indicates the plane selection mode, recommended or forced.
- scEmIncrease. Used to increase the switch from 10G to 20G. It also indicates the state of the increase activity.

3.1.4 The EM table

The Em table includes information about all EM:s except the SC EM. There is one row in the table for each EM. When a row is created the corresponding rows in the piu table are created automatically for the PIU:s belonging to the EM. This table holds information about:

- emSubrackId. The id of the subrack this EM belongs to.
- emSlotNo. The value of this object together with emSubrackId uniquely identifies the equipment module. The value should normally be the slot number of the first plug-in unit in this EM.
- emType. The type of the equipment module. When the equipment module is created the value decides how many plug-in units will be created and in which slots they may be placed.
- emAdmState. The adminastrative state of the EM.
- emOpState. The operational of the EM.

3.2 Equipment Interfaces MIB

The interface to this MIB makes it possible to perform block, deblock and test operations on the links in the system. There are two tables of interest in this MIB [7].

3.2.1 The ifTable

The ifTable contains standard information for all managed interfaces at a node. The rows in this table are created automatically when a row in the emTable is created, if this EM consist of PIU(s) that have links. The rows are deleted when the corresponding EM is deleted. Some useful entries in this table are:

- ifIndex. A unique value for each interface.
- ifName. The textual name of the interface.
- ifAdminStatus. The administrative state of the link.
- ifOperStatus. The operational state of the link.

3.2.2 The ifIndexToPhysTable

This table specifies the Subrack, Equipment Module and Plug-In unit slot number for each sublayer. Following entries are needed to

- ifIndexToPhysSubrackId. A reference to the associated subrack.
- ifIndexToPhysEmSlotNo. The slot number for the Plug-In unit that this link belong to.
- ifIndexToPhysEmPhy. The number of the physical link (on this specific EM).

3.3 The EQM Block Functionality

The functionality of the EQM block is implemented in Erlang Modules.

3.3.1 Eqm PIU Module

A PIU is a general term for the hardware units which are placed in a subrack. Within the subrack the PIU is placed in one equipment slot. This module contains one server for each PIU in the system. The PIU server represents one PIU in the subrack. On commands from the operator it is possible to:

- deblock a PIU server which then loads and starts the device processor on the real PIU. Thereafter it informs other software functions that the PIU is deblocked.
- block a PIU server which then stops the device processor on the real PIU and informs other software functions that the PIU is blocked.
- block a link on a ET-PIU server which then calls the device processor on the real PIU and thereby blocks the link in the hardware. Thereafter it informs other software functions that the link is blocked.
- deblock a link on a ET-PIU server which then calls the device processor on the real PIU and thereby deblocks the link in the hardware. Thereafter it informs other software functions that the link is blocked.

3.3.2 Eqm EM Module

An EM is the basic unit for configuration and extension of hardware resources and its related software resources. It consists of one or more Plug-in units. This module contains one server that is representing all EM:s in the system. On commands from the operator it is possible to:

- create an EM which then creates its PIU servers.
- deblock an EM which then deblocks its PIU servers.
- block an EM which then blocks its PIU servers.

- delete an EM which then deletes its PIU An EM is the basic unit for configuration and extension of
servers.

The ET and CP EM:s only have one PIU and the SC EM can have two PIU:s for a 10 Gbps configuration and four PIU:s for a 20 Gbps configuration.

4.0 Problem Definition

This chapter describes the present interface to control the hardware of the ATM switch. This will finally motivate the idea to make a GUI written in Java. A detailed list of the requirements of the GUI is given at the end of the chapter.

4.1 Present Equipment Management Interface

The equipment management functional area are split into three main areas: Create, Open and Delete (Fig.7.).

The Create operation menu provides a single operation menu title that allows you to perform the management operation to create a new EM. You select the operation menu title *Delete - Equipment Modules* to delete an EM. This opens a data list form which displays each EM that is configured to the AXD 301 system. In the Open operations menu you can perform management operations on a PIU, EM or a Link. You select the operation menu title *Plug-In Units* to perform block, deblock or test on a PIU. This opens a data list form which displays a table of all the PIUs configured to the AXD 301 switching system. Each row in the table summarises an individual PIU.

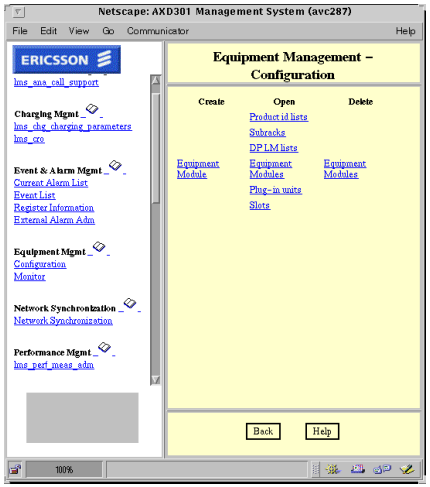

Fig.7. The equipment management configuration screen of the present control system.

You can access an individual PIU by selecting a table row icon. This opens a new data entry form where you can perform the desired management operation. If the PIU is a ET-CB, you can also select a hypertext linked value which shows a data entry of the links on the specific PIU. To perform any management operations on a link, you have to select a table row icon to open a new data entry form. From here you can perform

block, deblock and test on a single link. Operations corresponding to EMs are performed in similar way by selecting the *Open - Equipment Modules* menu title. All the html pages are created using scripts designed in Erlang. The Erlang application will function as a CGI script.

The problem with this interface is that it takes several steps through Erlang generated html pages to perform any management operations on the equipment. There exists some short-cuts through the menu system, but it still requires a lot of patient to wait for the pages to show up. It is possible to view the plug-in units and the links graphically but you can not perform any management operations on them.

Using Java to write the client would be very helpful. Java comes with an extensive set of classes, arranged in packages, that you can use in your programs. For example, Java provides classes that create graphical user interface components (the java.awt package) and classes to control the layout of components within their container objects. Java also provide classes to keep track of user and window system events (java.awt.event package). These arguments strongly motivate a client written in Java to view and operate the equipment of the ATM switch.

4.2 Detailed Requirements

Here follows a detailed list with the requirements of what actions the operator should be able to perform through the graphical user interface.

- An operator should be able to perform the management operations create, delete, block, deblock and test on an individual plug-in unit, equipment module or link.
- Keep track of the Administrative and Operative states of the equipment and display both to the operator.
- Operate the LEDs on every PIU in order to indicate the blocking state and if there are any alarms on the physical PIU.
- It must be possible to view both the front PIUs and back PIUs of the subrack.
- The interface must be automatically updated to display some dynamic data.
- It should be possible to get a logical view of the subrack where you can view and operate the EMs as graphical objects.
- The GUI should be as user friendly as possible.

In order to solve the requirements given, there must be some server processes to handle the requests from the client. These processes should be implemented in Erlang. The Erlang programs must provide functionality to:

- Receive requests from the GUI client and identify them.
- Provide functionality to create, delete, block, deblock and test on the hardware on request from the client.
- Interact with the hardware. Scanning the eqmMi and eqmIfMi for PIUs, EMs and Links and identify them.
- Send data to the java client to update the GUI.

The application should also be easily expanded in the future, both in terms of new equipment as well as for more program functionality.

5.0 The Architecture Of The Application

This chapter describes the Erlang modules and Java classes that is implemented to fulfil the requirements from chapter 4. The protocol designed to communicate between the client and server is also described.

The application is a combination of a client or a front end portion that interacts with the user and a server or back end portion that interact with the information resources within the AXD301 system. The front end is implemented in Java and the back end portion is implemented in Erlang. The client and the server processes interact through message passing mechanisms. The basic idea of the interaction between the client and the server is shown in Fig.8.

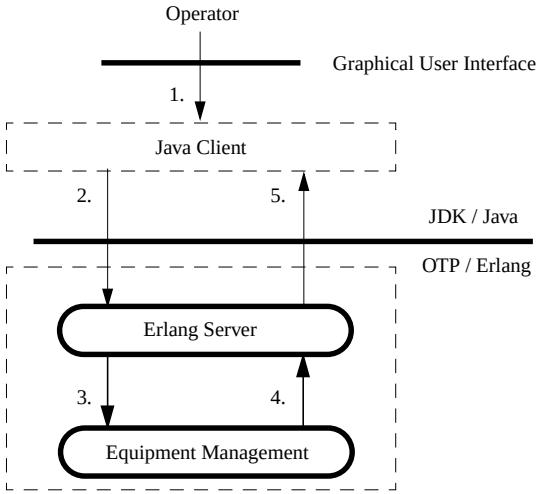

Fig.8.The Client Server Interaction.

The arrows in the figure has following meanings:

1. A user performs a management operation on a specific equipment unit through the user interface.
2. The Java client sends the requested operation to the Erlang server with necessary information to identify the equipment unit.
3. The Erlang server identifies the incoming request to call the appropriate instrumental function to a MIB.
4. The outcome of the operation is returned from the MIB. This will indicate if the requested action was successful or not.

5. A reply is sent back to the operator. If the action was successful the reply includes new data to update the interface. Otherwise the reply includes a message to indicate what went wrong.

The client runs in a thread that handles incoming network traffic. It acts as a dispatch for incoming packets. It decodes the packet and calls the appropriate code to handle the packet. Messages coming from the server can be treated just like any other event. The messages sent between the server and the client is an asynchronous operation so the client and the server will not wait for the message to either arrive at the destination or to be received. All message passing on the server side is synchronous, which means that the server must fully handle a request before it can receive another one.

5.1 Design

Most networked applications can be broken down into three basic modules [3]:

- The network interface module, that supports the socket communication between the client and the server.
- Application specific module, which provides functionality to perform all client requested operations.
- The user interface module, a set of java components that the user can interact with.

In order for these objects to work as a whole, they must have well-defined interrelationships. The user interface code should excel in handling user interface issues and should pass on any other actions to an appropriate program component.

5.1.1 Network Interface Module

This module integrate the Java and Erlang programming languages. In OTP there already exist an Erlang application which makes it possible for a Java Applet to communicate with an Erlang program. This application is called Jive [2]. Communication between the Client and Server is socket based and both Java and Erlang has Jive packages that hide the socket communication from our other code. Fig.9. shows the Jive architecture.

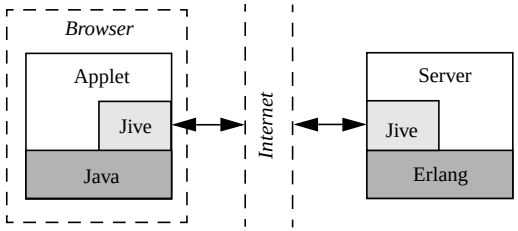

Fig.9.The Jive Architecture.

Before using any of the functionality in the Jive application a Jive server, which handles the interaction between Java and Erlang, has to be started. The Jive server listens to a pre-defined socket port through where it can send and receive messages. It acts like a message passer between the Java client and our Erlang code. Each message sent between the client and the Jive server contains one byte to describe the message type, and each message have a routine that knows how to parse the data. By using Jive, the Java client can communicate with Erlang through three mechanisms:

- Spawn new Erlang processes.
- Do Erlang apply on functions.
- Send messages to Erlang processes.

The Java side also contains wrapper classes for each Erlang variable type: integer, float, string, list, tuple, atom and pid. Once we have the underlying network model, we can start to design the client and the application specific server. Generally, they will have complementary methods. When one produces a message, the other knows how to parse it.

5.1.2 Project Specific Module

The core of the application is the Equipment management MIBs that contain all information about the equipment of the ATM switch. Both the client and server need access to this information. On the Client side, the information about the equipment is saved in memory, which will be updated within a certain interval.

The project specific module are designed in three Erlang modules:

- eqmGuiServer
- handleRequest
- submitRequest

The inter module communication is shown in Fig.10. Fig.10.The jiveServer listens to a predefined socket port. When new data is received, the message is parsed to identify the different messages that can be sent to the jive server, i.e. spawn, apply or send.

All ‘send’ messages are delivered to the eqmGuiServer in our application specific module. The eqmGuiServer receives all application specific requests:

- Get management data, which is used to fetch information from the MIB.
- Set management data, as a result to a create, delete, block, deblock or test command on a piece of equipment in the system.

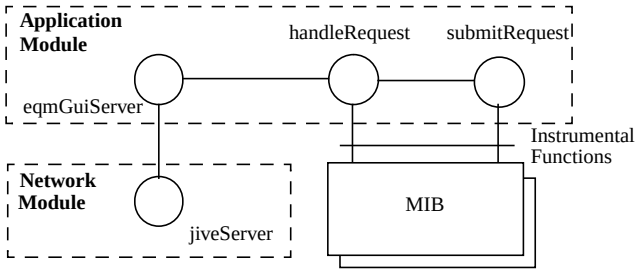

Fig.10. Inter-module calls on the server side

The eqmGuiServer is just a receiving process that parse incoming messages and invoke methods in the handleRequest module to perform the requested action. All requests to get management data are handled by the handleRequest module. All other requests to set management data due to a management operation is handled by the submitRequest module. The MIB, which holds the managed data, is called via the instrumental functions.

5.1.3 User Interface Module

At its best, designing a user interface can be an easy task using some tool to graphically lay out all your GUI components. At its worst, it involves laying out the design by hand. Screen layout in Java is accomplished by placing items on the screen by using some layout manager. Typically a combination of the layout managers provided with Java will accomplish what you want.

The GUI starts with one main screen that displays the front of the subrack with the currently configured plug-in units. From this screen, the user can choose from tasks such as block, deblock and test on a plug-in unit. It should also be possible to view the back of the subrack on demand from the operator. When the back of the subrack is shown, it is possible to perform management operations on the connection boards and the links on every ET CB.

From the main menu, the user can open a new window that shows the logical view of the subrack. This window is a separate frame that displays the switch core and the created EMs of the system. It is from this menu possible to perform management operations such as create, delete, block and deblock of a EM.

To represent the EMs and the physical hardware of the switch, the following classes have to be implemented:

- Subrack, which is the placeholder of the plug-in units.
- EM, which represents an equipment module.
- PIU, which represents a physical plug-in unit and is the place holder of the links in case of a ET CB.

- Link, which represents a link on a ET CB.

The EM, PIU and Link is implemented as custom components, which makes it possible to draw custom graphics on them and to receive input events from the user.

5.2 The Protocol of the Application

This section describes the protocol used to pass messages between the eqmGuiServer and the jive server (Fig.10.).

In Erlang the only form of communication between processes is by message passing [1]. A message is sent to another process by the primitive ‘!’ (**send**):

Pid ! Message

Pid is the identifier of the Erlang process to which **message** is sent. A message can be any valid Erlang term. Messages sent from the client are called request messages and messages sent from the server are called reply messages.

5.2.1 Request Messages

The request messages are those received by the eqmGuiServer. They are sent encapsulated within the Erlang variable type tuple of size 3:

{Request, Resource, Method}

The different fields are described below.

Request. The Request is an atom that identifies the type of the equipment, i.e. a plug-in unit, an equipment module or a link. The possible values for the atom in the Request field are:

- change_piu. This request means that the server should perform the method specified in the method identifier field upon the plug-in unit identified in resource identifier field.
- change_em. The change EM request means that the server should perform the method specified in the method identifier field upon the equipment module identified in the resource identifier field.
- change_link. This request makes the server to perform a management operation specified in the method identification field upon the link identified in the resource identifier field.
- get. The get request is used to fetch information from the MIBs. This request is used to automatically update the user interface.

Method. The method field contain an atom that identifies the management operation that should be performed on a piece of equipment. The different values in this field are:

- create. This method is used to create the equipment module specified in the resource field. It is only used together with a change EM request.

- delete. The delete method is used together with the change EM request to delete the equipment module specified in the resource field.
- block. The block method can be used together with any of the change requests. It makes the server to perform a block operation on the PIU, EM or Link specified in the resource field.
- deblock. This method is identical to the block method except it performs a deblock operation on the EM, PIU or Link.
- test. This method is used together with the change PIU or change Link requests to perform a test operation on the equipment specified in the resource field.

Resource. This field identifies exactly the piece of equipment to be controlled. It contains a List with the following information:

- SubId. This is an integer that makes a reference to a subrack.
- SlotNo. SlotNo is an integer between 1-20. It is used together with the SubId and Side to identify a slot in the subrack.
- EmNo. The EmNo is an integer that identifies an EM.
- Side. This is an integer that identifies the side of the subrack. If the integer is equal to 1, the referred side is front and if the integer is equal to 2 the referred side is the back of the subrack.
- EmType. An integer that specifies the type of an EM.
- Name. This Parameter is a string that gives a user friendly name to an equipment module. It is only meaningful together with a create operation.

5.2.2 Reply messages

After receiving and interpreting a request message, the server replies. The replies look like this:

{send, Receiver, Message}

send is an atom that tells the Jive server to forward the contents of the Message field to the Java client. **Receiver** is an integer that identify the Java object that should receive the message. **Message** is a List of variable size. Two types of reply messages:

Update. The update response is used by the server after a successful management operation or after the **get** request. the message include information from the tables in the MIBs that is used to update the user interface.

Textmessage. This makes the client to display a window with a specified textstring. The textmessages are used as a reason phrase of a previously request failure.

6.0 Implementation

This chapter describes the implementation of the application specific module or the server and the Java classes implemented for the front end. A manual for the GUI is given at the end of the chapter.

6.1 Server Implementation

The server part consists of three Erlang modules:

- eqmGuiServer. This module receives the requests from the client.
- handleRequest. The handleRequest validates the incomming messages to check if requested action is permitted. It provides functionality to fetch information from the tables in the MIBs.
- submitRequest. This module is invoked when a management operation has passed the validation algorithm in the handleRequest module.

6.1.1 eqmGuiServer

The jive module is as we already mentioned implemented in the OTP system. Before using any functionality in the jive module a jive server has to be started. This is performed by the init function in our eqmGuiServer:

```
init(Port) ->

    jive:start(Port),

    jive:allow({guiServer,start,2}).
```

Port contains the socket port that the jive server should listen to. The jive server is started in the next line of the code and makes it thereby possible to perform Erlang apply on functions and to send messages to an Erlang process. To prevent security problems it is only possible to access functions which are explicitly declared by the Erlang side. This is done by the allow call in the last row of the code above. The call to this function contain a tuple which names the module, the function and the arity of the function that the client is allowed to access. This means that the client is allowed to spawn function start in the Erlang module eqmGuiServer. The init() function is only called once, when the server process is started for the first time. The start method is spawned by the Java client when the user visits the html-page with the Applet. The code for the start function look like this:

```
start(Receiver, PidId) ->

    Pid = jive:get_pid(PidId),

    receiver(Receiver, Pid).
```

Receiver is an integer which specifies the Java object that should receive replies from the server. When the client connects to the Jive server, the server starts an Erlang process which allows the Jive server to send messages to it. The Java client knows this proc-

ess as the self process [2]. The PidId is an integer that identifies the Java client that should receive the information sent from the server. In the second row of the code the actual Pid of the Erlang process associated with the PidId is returned. The receive function is called with Receiver and Pid as parameters. This will initiate a server loop.

The eqmGuiSever is our application specific server which is a very simple loop. It receives the requests from the operator through the Jive server. It runs continuously waiting for requests from the client. Here is the code for the main server loop:

```
receiver(Receiver, Pid) ->

    receive

        {Request, Resource, Method} ->

            Result = handlerequest:handle_request(Request, Resource, Method),

            Pid ! {send, Receiver, Result};

        Other ->

            receiver(Receiver, Pid)

     end.
```

The eqmGuiServer process has its own mailbox and all messages which are sent to the process are stored in the mailbox in the same order as they arrive. The **receive** primitive is used to receive messages. In the above, {Request, Resource, Method} is a pattern, actually an Erlang tuple of size three, which are matched against the messages that are in the process's mailbox. The pattern is used to parse the message. When a matching message is found, the message is selected, removed from the mailbox and then another process, handleRequest, is called to handle the corresponding action. The variables in the pattern is at first unbound but becomes bound when a tuple of size three is selected from the mail box. Any messages which are in the mailbox and are not selected by **receive** remains in the mailbox and will be matched against the next **receive**. The last clause in the **receive** has the unbound variable **Other** as its message pattern. This will match any message which are not matched by the first clause. Here we ignore the message and continue by waiting for the next message. This is the standard technique for dealing with unknown messages: **receive** them to get them out of the mailbox.

6.1.2 handleRequest

The handleRequest module is invoked to perform all application specific requests. It has one function that is called by the eqmGuiServer to handle the requests:

```
handle_request(Request, Resource, Method) ->

    Result=apply(handleRequest, Request, [Resource, Method]).
```

The Request parameter is an atom. This atom names the function that is to handle the requested action. Resource is a list which identifies a piece of equipment and Method is an atom that specifies what to do with the Resource, i.e. block or deblock. apply is a

BIF which apply the function Request in the module handleRequest to the argument list [Resource, Method]. The handleRequest module defines following management functions to handle all requests:

change_piu. This function is invoked to handle all management operations on each plug-in unit. It checks the administrative and operational states of the PIU to validate if it still is in correct state. It should for example not be possible to perform a block operation on an already blocked PIU.

change_em. The change_em function is invoked to perform management operations on an equipment module. The state of the EM is checked to validate the operation.

change_link. Called to perform a management operation on a link.

get. The get function is called to fetch information from the MIBs. The data is sent to the Java client to update the interface.

Each management operation that is correct is finally used to set new values in a MIB table. The submitRequest module defines functions to change a table due to a management operation.

The Result from the apply call is returned to the eqmGuiServer which sends it to the client. The reply is either new data to update the GUI or an error message.

6.2 The Graphical User Interface

In this section, we only show the highlights of the user interface code. The most important classes of the GUI is examined in more detail. The components that is implemented to describe the equipment of the ATM switch works in a similar way, so we only describe the implementation of the PIU class.

Fig.11. shows the class hierarchy on the client side. It shows that the GUI components all inherit the Panel class and implement the ActionListener interface from the Java API package [8]. Panel is a container that does not have its own window - it is contained within some other container. The arrows in the figure shows the relations between the container components. The main container is the EqmGuiClient which holds contain the Subrack, PIU and Link components. The EM component is added to a separate frame.

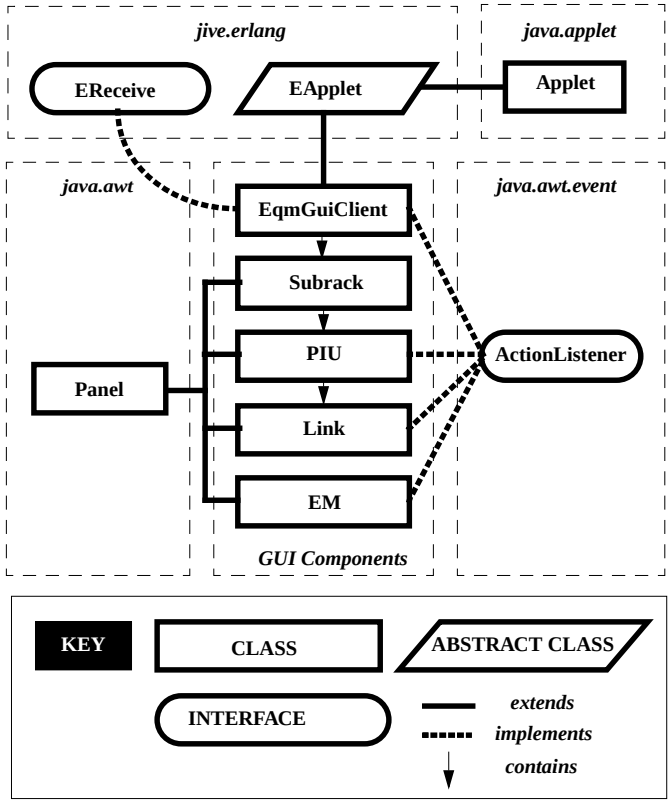

Fig.11. The class hierarchy of the Java client.

6.2.1 The EqmGuiClient class

The EqmGuiClient class is the main container of all the GUI components. It is a subclass of the EApplet class from the jive package. The EApplet overrides the init method in the Applet class. This method is called in the constructor of the EqmGuiClient to initialize the Jive client. This will take care of the connection with the server. The Jive package also provide the EReceive interface which must be implemented by each class which want to receive messages from the Erlang server. One method has to be implemented from the EReceive interface:

```
public void receive(EVar var) {

    EVar evars[] = ((EList)var).value();

    String guard = ((ESting)evars[0]).value();

    if(guard.equals("textmessage"))

        // display textmessage in a window

    if(guard.equals("update"))

        // update all GUI components

}
```

The receive method is invoked whenever some data are to be received from the eqmGuiServer. It is called with the message as parameter when the Erlang side wants to communicate with the object. EVar is an abstract class that is a superclass of all other Erlang wrapper classes. The var parameter should be of the Erlang variable type list, if following the instructions of the protocol. In line two of the code above, the whole message is parsed into an array of EVar objects. The first position in the array should contain an EString object identifying the type of the message. The EString object is converted into a Java string which is used in the if-statements to call the right functionality for handling the rest of the message. The first if statement is matched when the server has a textmessage for the EqmGuiClient. A window is then shown with the message. This happens when an error has occurred due to a management operation, and the server replies with an errormessage. The second if-statement is called as a reply to a successful management operation or as a reply to an update request. All custom components that represents the equipment of the ATM switch is then updated.

The EqmGuiClient defines methods that makes it possible for other Java objects to communicate with the Erlang side. For example, changePiu() is a public method that is used to send a block, deblock or test request to the Erlang server.

The constructor of the EqmGuiClient loads all images used by the interface. It uses a MediaTracker to fully load the images before they are displayed.

6.2.2 The Subrack class

The Subrack class is the container of the PIU components. It uses one Panel to arrange the PIUs corresponding to the front of the subrack and another Panel to arrange the PIUs corresponding to the back of the subrack. The Panels uses a FlowLayout manager to layout the PIU components within the Panels. The Subrack then uses a CardLayout manager to make just one Panel visible at a time:

```
setLayout(new CardLayout());

add((Component)frontPanel_,"FRONT");

add((Component)backPanel_,"BACK");
```

The frontPanel_ and the backPanel_ parameters is the Panel objects with all of the front and back PIUs of the subrack. "FRONT" and "BACK" are strings identifiers associated to the Panels and used for fast random access of the Panels. The Subrack defines a public method that makes it possible to flip between the back and front of the Subrack:

```
public void show(String side) {
    ((CardLayout)getLayout()).show(this,side);
}
```

This method is called with the desired side to be displayed as a parameter.

6.2.3 Creating a Custom Component

Three custom components are implemented to describe the switch hardware. They all inherits the Panel class which enables it to draw custom graphics on the Panel and to trap input events from the user.

The most important method of any custom component is the paint() method, which actually draws the component. The paint method in the PIU class look like this:

```
public void paint(Graphics g) {
    g.setColor(Color.white);
    g.fillRect(0, 0, getSize().width, getSize().height);
    if(!slotState.equals(“unusedEmpty”))
        // Draw images based on the type of the physical plug-in unit.
    else
        // Fill rectangle with color representing an empty slot.
}
```

The first thing that happens in the paint method is that the whole size of the component is filled with white color. There exists one PIU object for every slot position in the sub-rack whether the slot contains a physical plug-in unit or not. This is what the first if does, decide whether the PIU object represents a slot to which a physical plug-in unit is inserted. slotState is a private member that contains a string with the current state of the slot (3.1.2). If there is some hardware inserted into the slot, a PIU type specific image is drawn on the screen. The LEDs of the PIU is also drawn. The else-clause is matched if the slot is empty which just fills the rectangle with a dark color representing an empty slot. The slotState variable is also used to decide whether the plug-in unit has to be repainted or not. To avoid necessary processing time to repaint a component that hasn’t changed since the last time it was painted, the current slotState and the last slotState of the PIU object are compared. This decides if the component must be repainted or not: If the slot state has changed and the new state indicates that a physical plug-in unit is removed or inserted into the slot, the PIU component must be repainted. This will make the GUI to be updated much quicker and avoid flickering images during a update procedure.

Like all custom components it has to override the getPreferredSize() and the getMinimumSize() methods to display itself on the screen. All PIUs are of equal size.

Another important method in the PIU class is setPiuType(). This method take care of creating and adding the links to the PIU container in case of CB ET. A code fragment for this method is given below:

```
public void setPiuType(String piuType) {
    piuType_ = piuType;
    if(piuType_.equals("ET155-S1.1 back")) {
        if(links_ == null) {
            links_ = new Link[4];
            links_[0] = new Link(sLinkImages_,subrackId_,slotNo_,0,eqmGui_);
            links_[1] = new Link(sLinkImages_,subrackId_,slotNo_,1,eqmGui_);
            links_[2] = new Link(sLinkImages_,subrackId_,slotNo_,2,eqmGui_);
            links_[3] = new Link(sLinkImages_,subrackId_,slotNo_,3,eqmGui_);
            addLinks();
    }
}
// One if statement for every possible CB ET.
else
    if(links_ != null) {
        removeAll():
        links_ = null;
    }
}
```

There is one if statement for every possible type CB ET. In the example the PIU type is a “ET155-S1.1 back” which has four optical links. The second if-statement checks if the links already exists which means that the Link components are already laid out on the PIU container. In case this is a new PIU object the Links are created and stored in an array of Links. addLinks() is called to layout the Links. The PIU container uses the GridBagLayout manager to organize the link components as realistic as possible. The else statement is matched if there are no physical plug-in unit in the slot or if the PIU is a SC PIU or a CP-IO PIU. The link components are then removed and the array with the links are set to null. The Link components are therefore created and deleted dynamically.

6.3 Sources of Events

All of the custom components interact with the user in the same way. They are all containers with their own popupmenu added to them. The popup menus has one menuiteam for every possible action you can perform on the component. Below is a code fragment from the constructor of the PIU class:

```
pm = new PopupMenu("Slot - " + Integer.toString(getSlotNo()));

add(pm);

blockPiu_ = new MenuItem("block PIU");

blockPiu_.addActionListener(this);

pm.add(blockPiu_);

// one menuitem for ecah operation you can perform on the component
```

The first line creates the pop-up menu with a title. The title shows the slotposition of the PIU object you can operate through the menu. The next line adds the pop-up menu to the PIU component. A menuitem to perform a block operation is created in the third line and the PIU is added as the action listener for this menuitem. The menuitem is then added to the pop-up menu. Whenever the pop-up menu is shown and the "block PIU" menu choice is selected, an action event is delivered to the parent of the pop-up menu. In this case the PIU component.

The enableEvents() method is invoked to have a specified event delivered to the component regardless of whether or not a listener is registered. Each component also implement the ActionListener interface to be able to receive the action commands from the menuitems.

6.3.1 Handling Events

The user interact with the GUI through mouse operations. The processMouseEvent() is called by the system when the user presses a mouse button or moves it over the component. The code for this process is given below:

```
public void processMouseEvent(MouseEvent e) {

    if(e.isPopupTrigger())

        pm.show(e.getX(), e.getY());

}
```

The if statement checks whether this event is the popup-menu trigger event for the platform. The next line shows the popupmenu at the x,y position relative the origin component. All other mouse events are ignored by the component.

menus take care of displaying themselves, but attaching behaviour to menu items is something you must handle yourself. The first order of business is to identify which

menu item has been selected, and then to go about the implementing the functionality associated with the menu item. To identify which menu item that was selected we simply implement the actionPerformed method from ActionListener interface for receiving action events. Whenever an action event occurs on a menu item, an ActionEvent is dispatched to the GUI component. The code for handling action events in the PIU class is described below:

```
public void actionPerformed(ActionEvent e) {

    String command = e.getActionCommand();

    if(command.equals("block PIU")) {

        eqmGuiClient_.changePiu(getSubrackId(),getSlotNo(),

                                getEmNo(),getSide(),"block");

    return;

}

. . .
```

The menu item that fired the action event is identified by the getActionCommand(). command is then used in a sequence of if statements. There is one if statement for each menu item. The code shows the case when a block operation is performed on a PIU object. An instance of the EqmGuiClient is then used to call the public method changePiu() together with necessary information to identify the physical hardware of the switch. The EqmGuiClient take care of processing the operation to the server side.

6.4 User's Guide

Using the program is fairly simple. After the program is stared, the client connects to the server. The server return information from the tables in the MIBs and the client lays out its GUI components for the first time. Fig.12. shows the back view of the subrack with its currently configured plug-in units. The figure shows the CLK boards (slot 1 and 20) of the switch and a ET-CB of type 155 Mbps electrical (slot 4). The ET-CB has 4 links.

The back view enables management operations on the PIUs on the back of the subrack and the Links on every CB ET. To operate the PIUs on the front of the subrack you have to press the button labelled "Front". This flips the subrack around and the PIUs on the front of the subrack is shown.

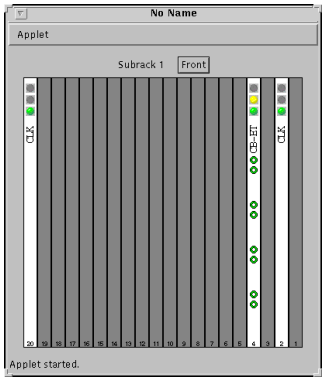

Fig.12. Back view of the subrack.

6.4.1 Inspecting and Modifying the Plug-In Units

Select the PIU you want to operate by pressing the right mouse button on it. This brings forward a pop-up menu that allows you to:

- Block the PIU if it is currently unblocked by selecting the “block PIU” menu option.
- Unblock the PIU if it is currently unblocked by selecting “deblock PIU” menu option.
- Test the PIU by selecting the “test PIU” menu option. The PIU has to be block in order to enable this operation.
- Inspect the PIU by selecting the “Info” menu option. This brings forward a window with the information about the plug-in unit.

6.4.2 Inspecting and Modifying the Links on a Plug-In Unit

Select the rear view of the subrack (Fig.12.). Select the Link you want to operate by pressing the right mouse button on the Link. This shows the pop-up menu which makes it possible to:

- Block the Link if it is currently unblocked.
- Deblock the Link if it is currently blocked.
- Test the Link.

6.4.3 Creating an Equipment Module

Press the right mouse button outside the subrack and select the “EM view” option. This brings forward a window (Fig.13.) with the currently configured equipment modules of the system.

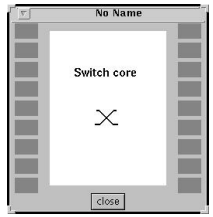

Fig.13. The EM view

The dark fields indicate the EMs you can create and the white fields indicate that a EM is already created. The only EM present in the figure is the SC EM.

Select the EM you want to create by pressing the right mouse button to visualize the pop-up menu and select “create EM”. This brings forward the window shown in figure (Fig.14.).

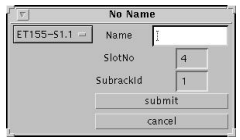

Fig.14. Create EM window

You select the EM you want to create from the pull down menu in the upper left corner of the window. You can also give the EM a user friendly name. To send the request to the server, press the submit button. To cancel the operation press cancel.

6.4.4 Inspecting and Modifying Equipment Modules

You can perform a management operation on an individual EM from the EM window (Fig.13.). Press the right mouse button on the EM you want to operate to visualize the pop-up menu. This allows you to:

- Block an equipment module that is currently unblocked by selecting the “block EM” menu choice.
- Unblock an EM whose adminastrative state is currently blocked by selecting the “deblock EM” menu choice.
- Delete an EM that is currently blocked.
- Inspect an EM by selecting the “info” menu choice. This brings forward a new window that displays information about the EM.

7.0 Conclusions and Improvements

All the requirements from the problem definition are fulfilled. The application works well and respond fast to user actions once it is initialized. By demonstration it takes longer time to start the Applet then to bring forward the main screen of the present CGI/Erlang user interface. After loaded, however, the Java interface is just about real-time to perform a management operation.

As stated in the problem definition the cgi interface required menu steps through Erlang generated html pages to perform any management operations. This is accomplished with only three different menus in the Java interface; one window makes it possible to view and operate the PIUs on the front and back of the subrack, and another window enables operation and monitoring of the equipment modules in the system.

Since a page in the html-interface is created dynamically it requires that the whole page is fully reloaded every time you visit it. The Java Applet on the other hand only loads the images to be drawn upon the GUI components once. After that, the only data passed between the applet and the server is new information about the switch hardware. This saves you considerable web server load.

7.1 Future Features

There are some future features that can be implemented compared to the present user interface:

- Perform an increase operation on the subrack, i.e. to upgrade 10 Gbps switch to 20Gbps.
- Modifying a EM name or PIU name by assigning a new value to the Name attribute. The name attribute can only be set at creation of an equipment module.
- Double check user operations. To prevent accidental operations it is desired to have a check window that asks for an acknowledgement of the requested operation and disables any other actions before the window is closed.
- Make a better error handling that tell the user more exactly what went wrong and what to do about it.

8.0 Abbrivations

API	Application Programming Interface
ATB	ATM Termination Board
ATM	Asynchronous Transfer Mode
CB	Connection Board
CGI	Common Gateway Interface
CP	Control Processor
EM	Equipment Module
EQM	Equipment Management
ET	Exchange Termination
GUI	Graphical User Interface
JDK	Java Development Kit
LCT	Local Craft Terminal
LED	Light Emitting Diode
MIB	Management Information Base
OTP	Open Telecom Platform
PID	Process Identifier
PIU	Plug-In Unit
SC	Switch Core
SNMP	Simple Network Management Protocol

9.0 References

[1] Armstrong, J., Virding, R., Wikström, C. and Williams, M., 1996, *Concurrent Programming in ERLANG*:707-750, ISBN 0-13-508301-X, UK: Prentice Hall International Limited.

[2] Grebenö, J., Nygren, K., 1997, *Jive Application (JIVE)*, http://otp.ericsson.se/product/otp_unix_r2d/lib/jive-1.0/doc/javadoc/Package-jive.erlang.html.

[3] Thomas, M.D., Patel, P.R., Hudson, A.D. and Ball JR., D.A., 1996, *Java Programming for the Internet*, ISBN 1.56604-355-7, US: Ventana Communications Group, Inc.

[4] 4/155 13-CNA 12165, *Equipment Management - AXD 301 ATM Switching System*, Ericsson Telecom AB 1997.

[5] 1/155 17-CRA 120 03, 1997, *Function Specification - Equipment Management*, Ericsson Telecom AB.

[6] 4/196 03-CRA 120 03, *Axd301Eqm-SWS.mib*, Ericsson Telecom AB.

[7] 6/196 03-CRA 120 03, *Axd301EqmIf-SWS.mib*, Ericsson Telecom AB.

[8] Package Index, http://java.sun.com/products/jdk/1.1/docs/api/packages.html.

10.0 Appendix A Constructor & Method Index