Diploma Project

Telecommunication equipment management using web services

HTML Service

Version V 1.1 (HTML)

Professeurs:
Philippe Joye
François Buntschu

Mandatory:
Daniel Gachet

Expert:
Nicolas Mayencourt

Students:

Thierry Kiki
David Schneider

Table of Contents

1 Definitions

MO:	Managed Object
MOM:	Managed Object Model
moType:	Managed Object Type
MF:	Management Function
ADF:	AccesPoint (MO) – definition file
MCST:	MileGate Configuration Software Tool
KOAP:	KEYMILE Object Access Protocol
SOAP:	Simple Object Access Protocol (W3C recommendation)
WebServices:	W3C recommendation
GUI:	Graphical User Interface

2 Initiation

2.1 Project Objective

Find a good way to generate on the fly HTML pages within the MilGate which is providing a web browser access.

2.2 Background

It would be interesting to offer a possibility to display and modify the configuration of the MileGate network device for humans. The most simple and standardized way is to provide the access via a web browser as a lot of other network devices as routers, modems, acces points or switches do.

Our only interface to access the data or configuration parameters is the MileGate Object Model with its proprietary communication protocol.

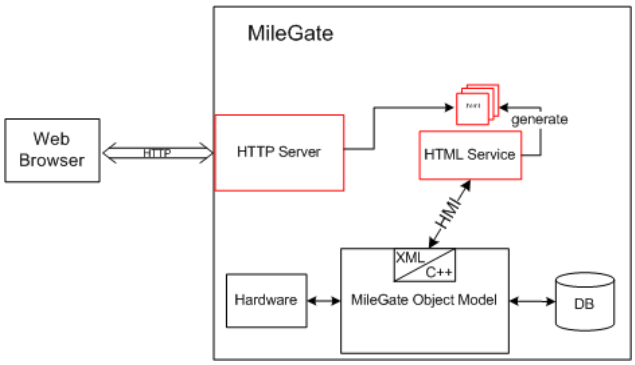


Illustration 1: System structure

For further treatment of the data for the presentation layer, we need to know the overall structure of the configuration (possible parameters) which needs to be parsed from an XML Schema, the ADF (proprietary AccessPoint Definition File) or in the future from the description of our Web Service (WSDL File).

2.3 Scope

2.3.1 Business functions

The aim is to analyze the feasibility of a service on the MileGate which creates HTML pages on the fly (run-time). It must be possible to change the configuration of the MileGate via an web browser.

It is not possible and not wished to to have the complete information in the memory because we would create redundancy which is complex to to manage.

It is imaginable to save the navigation structure on the system but all the data will be requested on use.

The service must be adaptable with a modular structure. Also the presentation layer and the logic must be separated strictly.

2.3.2 Project interfaces

The service will run on the MileGate core card (limitations will be discussed later) and is accessing the management interface. Information about the complete system are published in the document “User Guide – MileGate & MCST”.

For the implementation in C++, the document “C++ Programming Style Guidelines, Common Part” and “C++ Programming Practice Guidelines, Common Part” need to be respected.

2.3.3 Required analysis

Embedded System limits

- use of memory
- performance of system

Actual management system

- functions
- operational implementation

Constraints for MileGate

Identification of problems for implementation

2.4 Project constraints

2.4.1 Project dates

This task of the project is initially limited at 5 workdays. It is possible to resume some parts at an advanced project state.

2.4.2 Interproject dependencies

The task does not dependent on further work of our project but we can eventually identify common problems. The survey of the actual system and interface will help to understand the functioning and simplify future workings.

3 Analysis

3.1 Embedded System limits

3.1.1 Memory limitation

The memory of the MileGate is limited and has to be used with fully aware. The program itself need to be adapted to the MileGate coding rules.

If its necessary to add images or other graphical elements, they could be loaded over the Internet. HTML is pure text and does not use lot of memory.

Core Card: 128MB / 256MB of RAM
128MB Flash Memory (no hard disk drive)

3.1.2 Performance limitation

The actual management system (MCST) generates a lot of request towards the management interface. Amelioration is possible but not vital.

Performance limitations rather have to be considered at the implementation of the HTML Service due to the generation of the HTML files and its storage uses much more system resources.

CPU: PowerPC 603E (~400MHz)

3.2 Communication with the MileGate

3.2.1 Describe the Client-Server system used to communication with the MileGate

The communication with the management interface uses a proprietary XML protocol named KOAP which is transported over a proprietary message transport protocol (replaced in future by SOAP sent with HTTP/HTTPS)

It is a matter of a simple request-response system. The client is allowed to send request and the server (MileGate management interface) returns a response with the an indication whether the request was successful or had an error.

The KOAP protocol additionally offers the possibilities to send attachments.

All the services handling the configuration must access this management interface.

3.2.2 Describe the format of the requests and responses

The following paragraph shows how the transmitted message should look like.

The actual management interface accepts request which looks as followed:

```
<?xml version="1.0" encoding="utf-8"?>
<request version="1" seq="1" destAddr="/unit-1/port-1">
        <mdomain id="main">
                <operation seq="1" name="setLabel">
                        <Label>
                                <user>User1</user>
                                <service>Service1</service>
                                <description>Description1</description>
                        </Label>
                </operation>
        </mdomain>
</request>
```

Illustration 2: KOAP request

The request addresses the Management Object Type (MO Type) "/unit-1/port-1" and the Management Function (MF) "main". The called function is named setLabel and requires the shown XML formatting.

For the response we observe the response on the function getLabel because the function used just before won't deliver any content. The response looks as followed:

```
<?xml version="1.0" encoding="utf-8"?>
<response version="1" seq="1" destAddr="/unit-1/port-1">
        <mdomain id="main">
                <operation seq="1" name="getLabel">
                        <execution status="success"/>
                        <Label>
                                <user>User1</user>
                                <service>Service1</service>
                                <description>Description1</description>
                        </Label>
                </operation>
        </mdomain>
</request>
```

Illustration 3: KOAP response

The requests has the same parameters as the request. Additionally the tag <execution> with the parameter status="success" has been added into the tag <operation>. An unsuccessful response would contain the execution parameter status="proc_error".

Within the <operation> tag, the values just send before in the setLabel function were returned.

3.3 Object Model and the actual management system (MCST)

3.3.1 Analise the structure of the Object Model

An introduction to the MileGate Object Model MOM can be found in the document "Introduction to the MileGate XML Management Interface" under "Minimal introduction to the MOM".

The tree is built by Managed Objects (MO) in a hierarchical model.

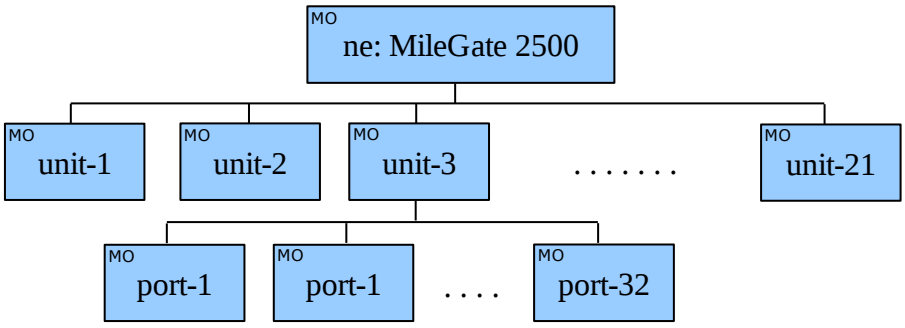


Illustration 4: MileGate Object Model structure

Each MO has its proper set of Management Functions (MF).

MF of root node: Main, Configuration, Fault Management, Status
MF of unit node: Main, Configuration, Fault Management, Status
MF of type port: Main, Configuration, Fault Management, Performance Management, Status

The complete structure of the **MileGate 2500** MF's is represented as annexe.

3.3.2 Analise the functional design

The first operation the MCST needs to know what type of equipment we are about to connect. Therefor a **Discover** message has to be sent to the root node's main management function.

```
<info>
   <moType>ne.milegate.2500</moType>
   <adfReference>keyne_r2e05pr_ws</adfReference>
   <addressFragment>ne</addressFragment>
   <moName>MileGate 2500</moName>
   <assignedMoName/>
   <state>ok</state>
   <adminState>na</adminState>
   <label>
      <user/>
      <service/>
      <description/>
   </label>
   <uuid>
      <id/>
   </uuid>
   <maxApAlarmSeverity>minor</maxApAlarmSeverity>
   <maxPropagatedAlarmSeverity>warning</maxPropagatedAlarmSeverity>
   <lastConfigChangedSeqNr>0</lastConfigChangedSeqNr>
   <lastSavedSeqNr>0</lastSavedSeqNr>
   <configChangedByLoad>false</configChangedByLoad>
   <hasChildren>true</hasChildren>
   <koapVersion>1</koapVersion>
   <eqpStatus>unprotected</eqpStatus>
</info>
.
<ChildList>

   .
   .
```

```
            <info>
               <moType>unit.holder.extended</moType>
               <adfReference>keyne_r2e05pr_ws</adfReference>
               <addressFragment>/unit-11</addressFragment>
               <moName>COGE1 R1D</moName>
               <assignedMoName>keyne_r2e05pr_ws</assignedMoName>
               <state>plugged</state>
               <adminState>down</adminState>
               <label>
                  <user/>
                  <service/>
                  <description/>
               </label>
               <uuid>
                  <id/>
               </uuid>
               <maxApAlarmSeverity>warning</maxApAlarmSeverity>
               <maxPropagatedAlarmSeverity>cleared</maxPropagatedAlarmSeverity>
               <lastConfigChangedSeqNr>0</lastConfigChangedSeqNr>
               <lastSavedSeqNr>0</lastSavedSeqNr>
               <configChangedByLoad>true</configChangedByLoad>
               <hasChildren>true</hasChildren>
               <koapVersion>2</koapVersion>
               <eqpStatus>unprotected</eqpStatus>
            </info>
            .
```

Illustration 5: Response of Discover

Additionally to the information about the equipment, the discover request provides a list of its children.

The complete structure of the Managed Object Type (MOType) **ne.milegate.2500** can be looked up in the AccessPoint Description File (ADF).

The GUI of the actual configuration tool MCST is generated automatically by parsing this ADF file.

```
      <mf name="main">
         <group name="general" cli="General" gui="General">
            <property name="Label" cli="Labels" gui="Labels">
               <struct name="Label" cli="Labels" gui="Labels">
                  <value name="user" type="string" range="63" gui="Label 1"/>
                  <value name="service" type="string" range="63" gui="Label 2"/>
                  <value name="description" type="string" range="127"
                      gui="Description"/>
```

```
            </struct>
         </property>
         <property name="AlarmSeverity" gui="Alarm Status">
            <struct name="AlarmSeverity" gui="Alarm Status">
               <enum name="maxAlarmSeverity" gui="Highest Alarm Severity">
                  <symbol name="cleared"gui="Cleared"/>
                  <symbol name="indeterminate" gui="Indeterminate"/>
                  <symbol name="warning" gui="Warning"/>
                  <symbol name="minor" gui="Minor"/>
                  <symbol name="major" gui="Major"/>
                  <symbol name="critical" gui="Critical"/>
               </enum>
               <enum name="maxPropagatedAlarmSeverity" gui="Highest
Propagated Alarm Severity">
                  <symbol name="cleared" gui="Cleared" helpText=""/>
                  <symbol name="indeterminate" gui="Indeterminate"/>
                  <symbol name="warning" gui="Warning"/>
                  <symbol name="minor" gui="Minor"/>
                  <symbol name="major" gui="Major"/>
                  <symbol name="critical" gui="Critical"/>
               </enum>
            </struct>
         </property>
      </group>
```

Illustration 6: ADF structure

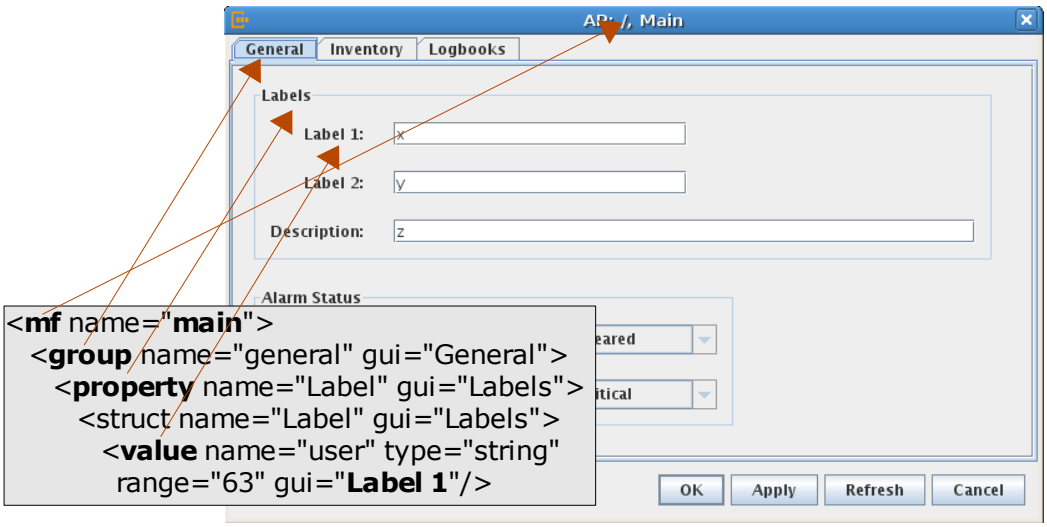


Illustration 7: MCST window

The structure is complete now, but the values are still missing. To get the values we need to send the proprietary KOAP message we mentioned before to the desired node.

To make the link between the ADF file, the KOAP message and the automatical generated GUI we represent once again the response for the getLabel function. The request was directed to the Management Function (MF) **main** with the property **Label** and the action **GET**.

```
<Label>
   <user>x</user>                  // ADF: <value name="user" gui="Label 1"/>
   <service>y</service>            // ADF: <value name="service" gui="Label 2"/>
   <description>z</description>    // ADF: <value name="description" gui="Label 3"/>
</Label>
```

Illustration 8: MileGate Object Model structure

3.3.3 Analise the dynamic adaptation mechanism

The MCST loads the complete tree of Managed Objects at the opening of the Application or if the user clicks on the refresh button.

If the user presses the refresh button, the entire management function MF (for example: main or configuration) will be regenerated with all its KOAP requests. This technique has the disadvantage that if we change just the timezone (MF: configuration), 41 KOAP requests need to be generated, sent and answered.

Those requests are generated very fast and it does not use much CPU usage to response them. But if we need to generate 15 new HTML pages (assumption that the structure of the GUI won't be changed) for each changing in this configuration management function, we waste lot of resources.

At the implementation of the HTML Service, we have to consider that we just send KOAP messages for the parameters which have changed.

3.4 Other constraints for MileGate

To be completed!

4 Feasibility studies

4.1 Identify problems for implementation

Problem	Description	Mitigation
Parsing HTML	It is difficult to extract information from HTML pages as it doesn't have a well defined structure.	The parsing of XML is much easier in C++. It could be a good solution to use XHTML instead of HTML.
Memory limitation for complete database. We can either create a DB for the service or always request the wanted parameters.	DB: + must get just modified parameter for regeneration - memory Direct output: + simpler to implementation - content of entire page must be requested on each modification	Creation of a DB probably won't be necessary for this implementation. We create additional problems caused by duplicating the data. Likewise is the implemented SAX parser on MileGate not optimal for the creation of a DB. In my opinion it's better to keep the number of request as small as possible.
Menu structure	The menu is complex but needs to be well arranged at the same time.	A good technique to use would be a solution based a tree menu (example JavaScript) for the navigation within the nodes and kind of pop-up menu for the management functions.
Refresh of navigation menu on insertion of new unit	The menu must be updated (rewrite HTML page) if a new unit is inserted. On the browser it can be reloaded automatically with a refresh timing.	We need to detect the low-level interrupt! To find the accurate method the survey of the MCST will be helpful.
File transfer on HTML	The actual system initiates file transfer with a tag and adds the file just behind. This is possible due to the protocol is no standardized.	Here we have to study how to use HTTP/Put in C++
Acknowledge on modification	If the user modifies a parameter, he needs to be sure that the operation was successful.	We can not send messages to the user with HTML (HTTP Server is between service and client). The only possibility is to print error messages on the HTML page

		which will be visible on the next reload.
Config of multiple Managed Objects (MO's)	The MCST GUI offers the possibility of configuring multiple MO's with one action.	This is difficult to implement in HTML, the task needs further studies.
Connection Manager (access the node)	The MCST GUI offers a connection manager which is user dependent.	The connection parameters of the users can not be managed trough the server. It is possible to use cookies to save connection parameters on the users web browser.
Customizing the GUI / Custom toolbar	A helpful add-on of the MCST is the customizable interface.	It will be very challenging to implement a customizable HTML page. The feasibility and its advantages should be studied in a further task. A custom toolbar is rather conceivable. It must also be saved on the client machine with a technologie such as cookies.
Printing option / Table CSV export	The MCST GUI offers a printing option and table export possibilities for spreadsheet programs.	Printing in HTML is obtainable with a well formated page or a additional stylesheet. The export possibility is more difficult and probably not supported in HTML. The CSV files may need to be generated within our HTML Service instead.

4.2 Description of a possible implementation

We want a product which is as modular and adaptable as possible. To achieve this we certainly need a strict separation between logic and presentation.

logic:

- contains parsing of structure and management of data (get/set via MOM interface)
- parsing of the object model
- interface between HTML and KOAP
- detection of changes

- new card: → parse
- modified config: → what to add/modify

presentation:

- contains presentation of data and generation/modification of navigation
- the data need to be represented in function of its usability/type
- the navigation need to be generated automatically
- Plug/unplug must modify the navigation according to the logic

5 Recommendation for Implementation

This topic contains our recommendation for the implementation of the HTML Service and an example user interface. The recommendations are based on the prior studies and converge in the basic structure towards the actual management system. This was necessary because no deep study on the structure was performed and with this, no change can be recommended.

5.1 Use Case diagram

The following diagram is a first approach to describe a possible functionality of the system.

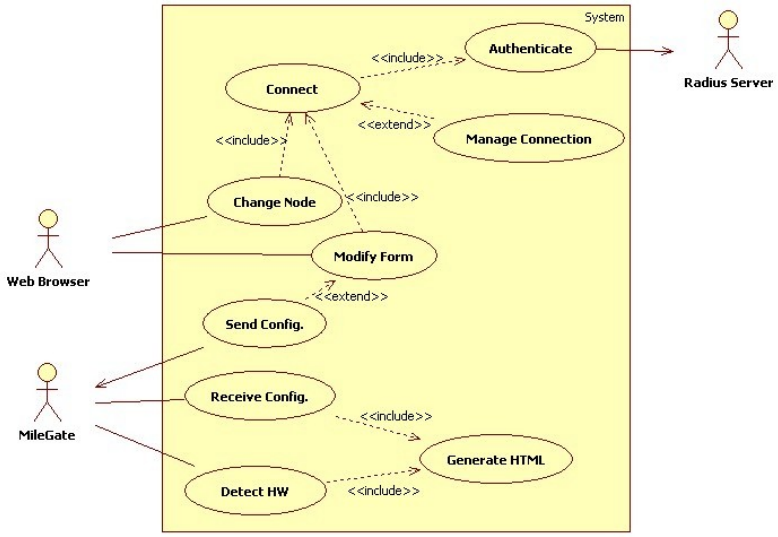


Illustration 9: Use Case diagram

5.2 Sequence Diagram

With the sequence diagram we like to show the sequential interactions and exchange of messages.

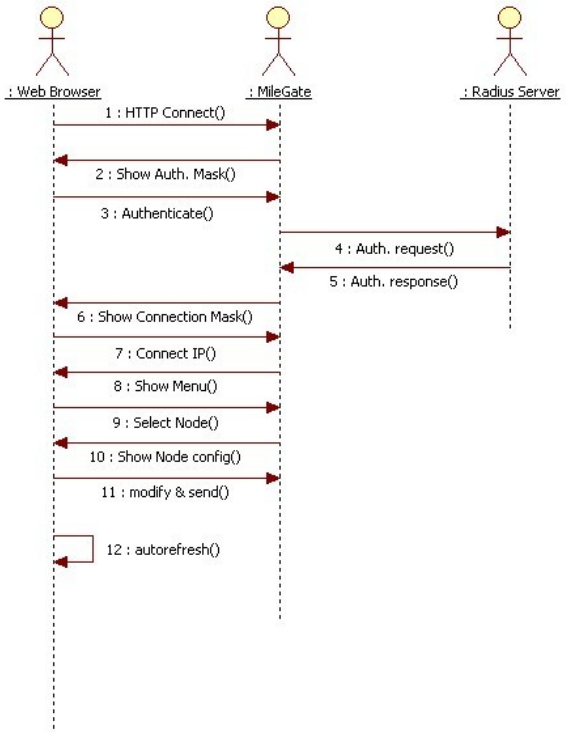
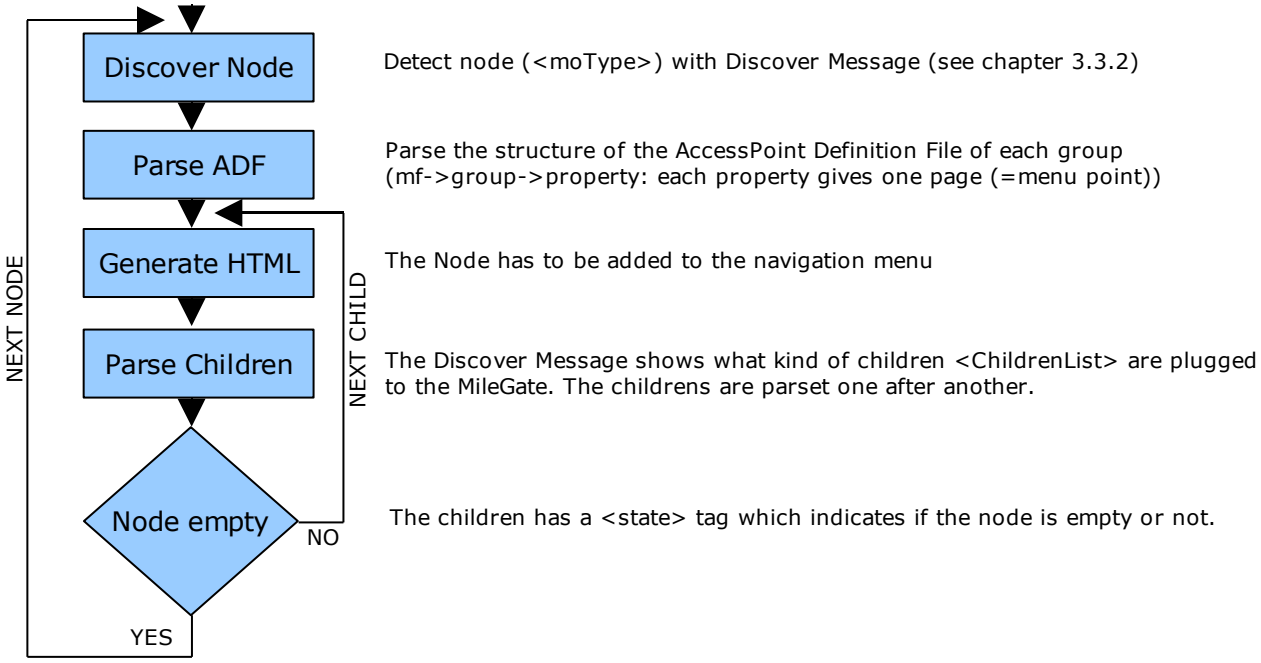


Illustration 10: Sequence Diagram

5.3 Operation of the HTML Service

At the initiation of the HTML service, the entire navigation structure has to be generated. The result of this will be accessible by the client after the step 7 of the Sequence Diagram. The connection itself does not evoke the initiation of the service, the structure needs to existing already at this point of time.

The following points visualize the basic functionality of the service and describe how the service can figure out the structure of the node.



It has to be said that the parsing of objects has to be recursive which is not represented in this flowchart.

5.4 GUI Prototype

The menu is the most important part of the website because it defines the way we can navigate trough the sites and with this the ease of use. Basically we have the root node with its units and ports. Further a technique need to be evaluated to add maximal five additional menus to access the further navigation structure (Main, Configuration Management, Fault Management, Performance Management and Status) of each node. Possibilities are a second navigation frame or a pop-up accessible by the right mouse button.

The following illustration provides a GUI prototype with a second navigation frame and pull down menus.

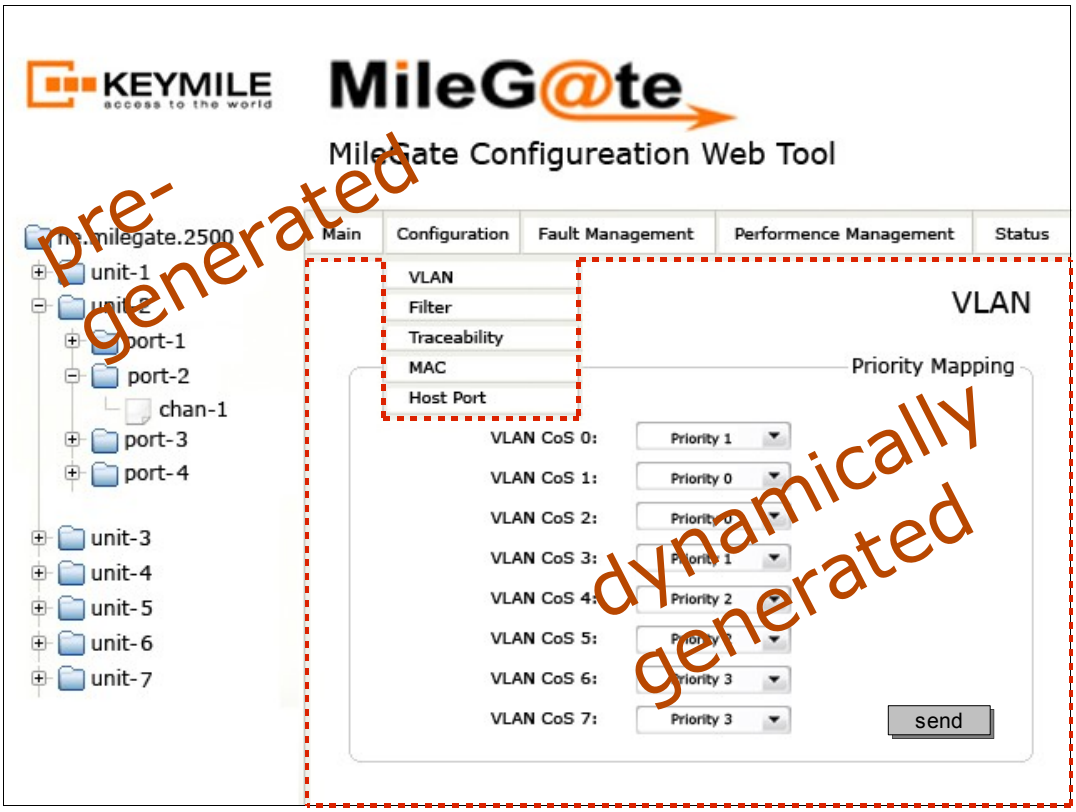


Illustration 11: GUI prototype

The website needs to be built with frames. That way we can use one single menu (left) on every other page. In function of the selection on the left side, the top menu and its menu points (top) need to change. As described before, the structure of this menu is defined in the ADF file for any possible kind of node.

If the user clicks on a navigation point, the real task for the service has to be performed. As we do not want to save the pages with the parameters anywere, we have to generate the entire content (exclusive of menus) at this point of time.

Request of content:

1. The possible form fields, check boxes, combo boxes, tables or buttons of one content frame is defined in the ADF file.
2. Transformation between ADF XML and HTML/XHTML has to be performed
3. To get the values we have to send KOAP messages with indication which parameters we would like (in the example case it would be:
request destAddr=”/”, mdomain id=”cfgm”, operation name=”getPriorityMapping”)
4. The service needs to merge the XHTML code and the parameters
5. Finally the XHTML has to be saved on memory
6. With a proper configuration of the HTTP Server, the file is now accessible by the user

5.5 Generation of HTML files

The generation of the HTML files is similar to the transformation already used for the JAVA client MCST. This is just the case if we decide to keep the actual structure described in the ADF file (see “Analise the functional design”).

ADF structure: <group><property><struct> or <group><property><table>

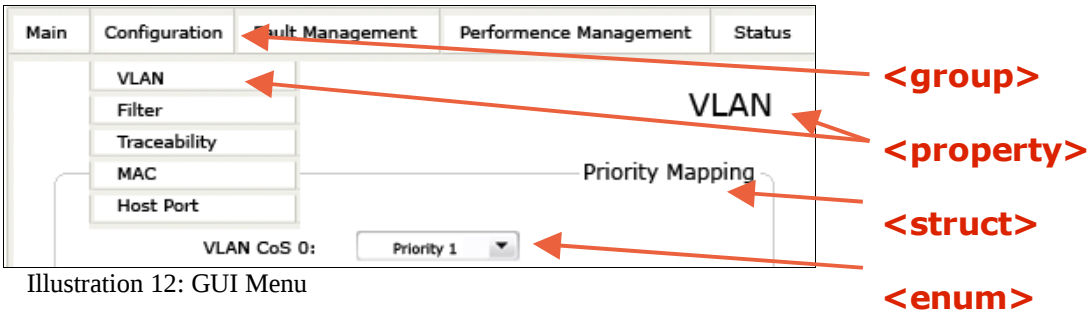


Illustration 12: GUI Menu

<group> → MENU Points

<property> → Page

<struct>/<table> → Content Element (multiple per page)

Within the <struct>, the tags could be transformed as followed:

<enum> : Combo Box

<value> : Text/Form Field

The transformation of the <table> tag is more complex.

5.6 Reaction on modification

The system needs to react to modification automatically. Modifications are possible on different interfaces such as CLI, MCST, syslog, SNMP and of course the web interface for this service.

The MileGate generates notification on a change of the configuration. Those notifications need to be captured by our service and must generate the new navigation automatically. This needs to be considered at the conception of the navigation structure. Those changes also complicates the automatic generation of the navigation structure.

5.7 Reaction on new Hardware

If a new unit is added or removed, the node needs to be added in the navigation menu and of course also deleted.

On addition of a unit a similar mechanism as the one described in the topic “Operation of the HTML service” has to be performed starting at the added unit instead of the root node.

5.8 Problems

Additional to the identification of the problems in the point “Feasibility studies” we list here some very important points for the implementation of the service.

Error Handling:

To announce errors to the user, we can just use the output of the HTML page. It is possible to generate error pages or to add the error message at any place of the page. We need to define what will be shown during the generation process to give the best feedback to the user.

Concurrent Problems:

The handling of concurrent access need to be checked to guarantee the functionality. In some cases, the access or the files have to be locked for secondary users.

Refresh Problems:

Automatic refresh of the HTML page with a refresh delay could cause some problems. We also have to pay attention that the caching mechanism of the browser/website is configured well. We need a very quick refresh time to not confuse/irritate the user.

Performance Problems:

We saw in the analysis that the embedded system has some limitations such as the performance. To avoid performance problems, proper testing is necessary.

6 Conclusion

A service which generates HTML pages on the MileGate is feasible.

The aim (advantage compared to MCST) and the wanted functions of such a service need to be planed and analyzed carefully. Main advantage is that a client does not need to install anything. It is also imaginable that browsers on mobile devices can be used.

The diagrams and process descriptions have no deep complexity and need to be expanded. Primary aim of those presentations was to provide an overview of the conceived approach.

At my point of view, a customizable user interface or a change of the look-and-feel could bring some advantages for the use of the interface.

The required time to realize this project is very difficult to estimate at the actual state because it depends heavily on the desired functionalities.

The survey of the actual management tool (MCST) and its implementation helped a lot and completed the introduction into the very complex MileGate system. We feel certain that this analysis will facilitate future tasks and helps if such a service will be designed.

7 Annexes

7.1 Revision history

Revision			Short description of the modification	Prepared by	Checked by	Approved
Doc ID	Version	Date				
HTML	1	09/06/09	First Version	DSCHN		
HTML	1.1	15/06/09	Modification after discussion on project session. (Major changes on chapters 2.3.1, 5, 5.3, 5.4, 5.6)	DSCHN		

7.2 References

- User Guide – MileGate & MCST
- C++ Programming Style Guidelines, Common Part (KEYMILE Confidential)
- C++ Programming Practice Guidelines, Common Part (KEYMILE Confidential)
- Extensible HyperText Markup Language http://www.w3.org/TR/xhtml1/

7.3 Structure of the MileGate 2500 Management Functions

The Structure can vary on different MileGate models, Units and Port types!

RootNode (MileGate 2500)

- Main
  - General
    - Lables: — Text fields
      - AlarmStatus: — Combo boxes
  - Investory
    - Equipment Inventory: — Text fields
  - Equipment
    - Database Compatibility List: — Text fields
      - Database Compatibility Function: — Table
  - Logbooks: — Buttons
- Configuration
  - Date And Time
    - SNTP Client
      - Operation Mode: — Combo boxes
      - Primary Server: — Text field
      - Secondary Server: — Text field
      - Polling Interval: — Text field
      - Broadcast Delay: — Text field
    - Time Zone: — Combo/Checkbox
  - IPSec
    - IPSec Parameters: — Checkbox
      - IPSec Policy: — Table
      - Internet Key Exchange: — Table
      - Security Methods: — Table
  - QoS
    - 802.1p Priority Mapping
      - 802.1p Priority – To – Queue Mapping: — Table
  - SNMP
    - General: — Button
      - SNMP Support: — Checkbox
      - SNMP Parameters: — Text fields
  - Syslog Destinations
    - Destination 1
      - Destination Configuration: — Text fields
        - Source: — Table

.

.

Destination 10:

Destination Configuration: Text fields

Source: Table

Syslog Source

Syslog Source List: Table

Temperature Limits

Exceed: Text fields

Warning: Text fields

ESO

ESO-1 Clock Sources: Check boxes

ESO-2: Check boxes

PETS

PDH Clock Sources: Combo boxes

PETS Clock Priority: Combo boxes

PETS Extraction: Combo box

Modify Password: Button

Configuration Management: Buttons

Configuration Status: Combo/Table

Management Interface

Management Interface: Text fields

Management VLAN: Text field

Packet

Bridge: Combo boxes

Traceability: Text field

PPPoA: Text fields

Session Management

Retry Time: Text field

Sessionmanager Session Timeout: Text field

Authentication Management Interface: Check boxes

RADIUS Default Userclass: Combo box

Radius Client

Radius Common Parameters: Check/ text fields

Primary Radius Server: Check/ text fields

Alternate Radius Server: Check/ text fields

Fault Management

Active Failures

Failure: Table

Status: Button

Alarm Status: Table

Configuration

Alarm Configuration: Table

Status

Radius Client
- Primary Radius Server Status: Combo/ text fields
- Alternate Radiius Server Status: Combo/ text fields

IPSec
- SA Status: Table

Redundancy: Buttons
- NE Configuration Status
  - Overall Configuration Status: Check box
  - Detailed Configuration Status: Combo/ Check box
- Core Unit Roles: Button/ text field
- Core Unit Status: Check boxes

Temperature: Button
- Current: Text field
- Max.: Text field
- Min.: Text field

ESO
- ESO-1: Combo box
- ESO-2: Combo box

PETS: Button
- PETS Clock Sourcfes: Check Table
- PETS Status: Combo box

Date And Time
- Time: Button
  - Summary: Combo/ text fields
- SNTP Client
  - Primary Server Status: Combo box
  - Secondary Server Status: Combo box
  - Last Response Time: Text fields
  - Last Jump Time: Text fields
  - Last Adjustment Time: Text fields

Management Interface: Button
- SSH Fingerprint: Text fields

Session Management
- Session: Table

Unit (SUAD1)

Main

General

Lables: Text fields

AlarmStatus: Combo boxes

Equipment: Buttons

Assignment Status: Combo/ text field

Equipment Status: Combo/ text fields

System Compatibility List: Text field

System Compatibility Functions: Table

Database Compatibility List: Text fields

Database Compatibility Functions: Table

Inventory

Equipment Inventory: Text fields

Internal Logbooks: Button

Cpuload: Buttons

File Access

Transfer: Button

Delete: Buttons

Software

Software On Element Manager: Table

Software On Unit: Table

Disk Space: Text fields

Configuration: Combo/ text fields

Configuration

VLAN

Priority Mapping: Combo boxes

Filter

Security Filtering: Combo boxes

Traceability

Logon Options

DHCP: Combo box

Aging: Combo/ text field

PPPoE: Combo box

MAC

MAC Options: Text field

Host Port

Policer Profile: Button/ Combo box

Fault Management

Status: Button

Alarm Status: Table

Configuration: Table

Commands: Buttons

Port (ADSL on SUAD1)

- Main
  - General
    - Lables: Text boxes
    - Alarm Status: Combo boxes
  - Admin And Oper Status
    - Administrative Status: Combo box
    - Operational Staus: Combo box
- Configuration
  - Multicast: Button/ Check boxes
    - Maximum Number Of Multicast Streams
    - Group Management
      - Multicast Access Profile 1: Button/ Checkbox
      - Multicast Access Profile 2: Button/ Checkbox
      - Multicast Access Profile 3: Button/ Checkbox
  - Traceability
    - Agent Remote ID: Text field
  - Security
    - Maximum Number Of Addresses for TLS: Text field
    - Maximum Number Of MAC Addresses For Nto1: Text field
  - Access Control
    - Classification Key: Combo box
  - Rate Limiter
    - Rate Limiter
      - Upstream Profile: Button/Combo/Check
      - Downstream Profile: Button/Combo/Check
  - QoS
    - Scheduling Profile: Button/Combo
  - Profile
    - Port Profile: Button/Combo
- Fault Management
  - Status: Button
    - Alarm Status: Table
  - Configuration
    - Alarm Configuration: Check Table/Button
- Performance Management
  - Performance Monitoring: Check boxes/Buttons
    - UserCounter: Table
    - History 15min Table
    - History 24h Table
- Status
  - General

DSL Status:	Combo box
Attainable Rate:	Text fields
Output Power:	Text fields
Band Table:	Table
Statistics	
Port Counters:	Text fields
Policing Counter:	Text fields
MAC Access Dynamic List	
Unicast List:	Table
MAC Forwarding List:	Button
MAC Forwarding List:	Table
QoS	
Weighted Fair Queues:	Text field
Multicast	
Stream:	Button
Active Streams:	Table/ Text field
VLAN	
Attached VLANs:	Table/ Text field
Defects	
Defects	
Central Office:	Check boxes
CPE:	Check boxes
Line Inventory	
DSL Vendor	
Central Office:	Text fields
CPE:	Text fields
Maintenance	
DSL Operation Status:	Combo box
RFI Bands	
Notch Table Downstream:	Table