Diploma Project

Manage Telecommunication equipment using Web Services

Acronym:	TELECOM-WS
Number:	D09T02
Date:	25.05.09 – 10.07.09

Professeurs:
Philippe Joye
François Buntschu

Students:
Thierry Kiki
David Schneider

Mandatory:
Daniel Gachet

Expert:
Nicolas Mayencourt

Table of contents

1 Introduction

Abstract

This first chapter introduces you into the Bachelor Project of Thierry Kiki and David Schneider. Necessary definitions and explications for the understanding of the report is provided here.

1.1 Definitions

MileGate

MO:	Managed Object
MOM:	Managed Object Model
moType:	Managed Object Type
MF:	Management Function
ADF:	AccesPoint (MO) – definition file
SFD:	
MCST:	MileGate Configuration Software Tool
KOAP:	KEYMILE Object Access Protocol

Web Service

XML:	Extensible Markup Language (W3C recommendation)
SOAP:	Simple Object Access Protocol (W3C recommendation)
Web Services:	W3C recommendation
GUI:	Graphical User Interface

Others

ECLI:	Embedded Comand Line Interface
HMI:	Human-Machine Interface
MMI:	Machine-Machine Interface

1.2 Project

This section describes the project and introduces the equipment. We defined the objectives we planed to reach at the initiation phase of the project. The distribution of tasks between the participators of the project will also be defined here.

1.2.1 Introduction

The company KEYMILE wishes a utility to manage its next generation telecommunication equipment with a system using web services. Actually, the management of the object model is performed either over an embedded command line (ECLI), syslog, SNMP or with the exchange of proprietary XML messages (KM-KOAP). The aim of this project is to find standardized solutions using web services (MMI) or to offer access via a web browser (HMI).

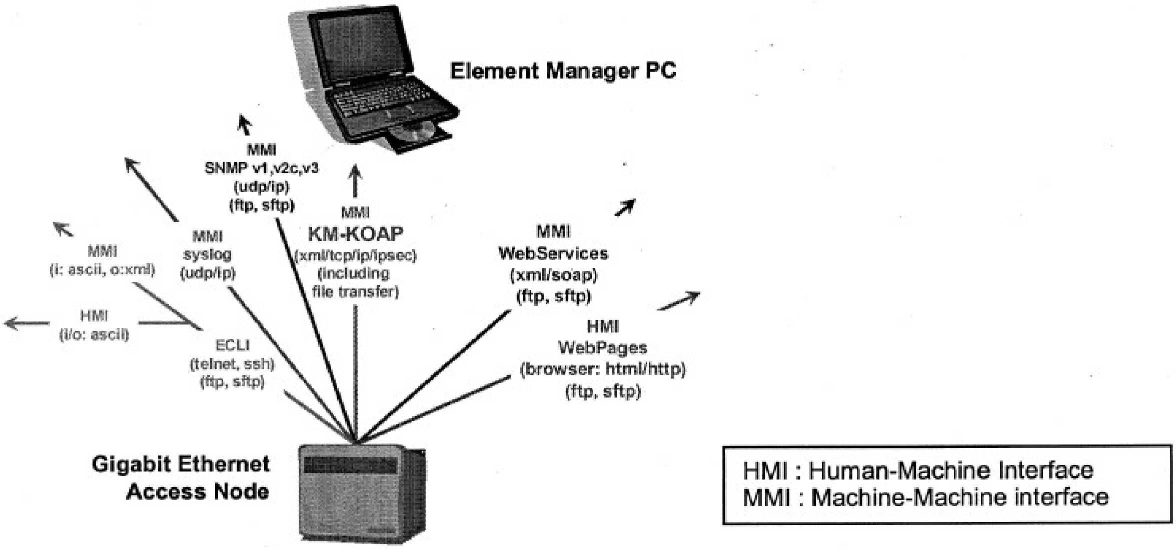


Illustration 1: MILEGATE management interfaces

1.2.2 Description of project

As this project needs to be adapted to the existing system, we need to respect a few constraints.
In the following image, the relations between the KEYMILE file describing the internal object model (SFD, XML) and the AccessPoint Definition File ADF (proprietary, XML).

1.2.2.1 Actual State

At the moment, the management system uses ADF which is a collection of SFD and describes one single unit in the MileGate. The core unit and access point of the MileGate is in slot 11.

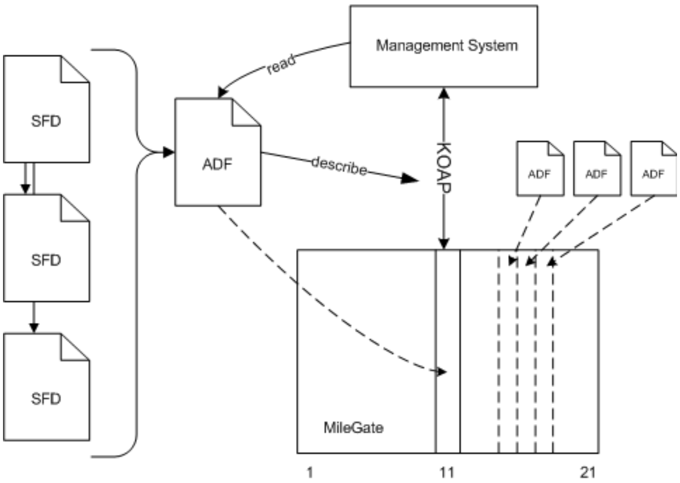


Illustration 2: Existing system

If we represent the actual communication more in detail, we see how the existing management utilities access the MileGate Object Model.

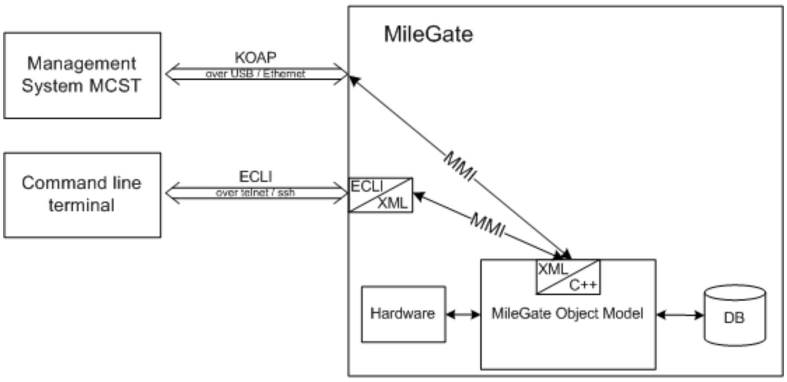


Illustration 3: Existing access methods

1.2.2.2 Work to perform

We have two new approaches for accessing the MileGate Object Model. The tasks to perform are represented in red.

1.2.2.2.1 Machine-Machine Interface (MMI)

The Web Service Description (WSDL file) which would finally be created will be the input for the client framework. The framework will generate code (for example Java, C, C++, Perl, Pyton, PHP, ..) automatically according to the constraints defined in the web service description.
The messages of the type SOAP (transported over HTTP) are treated within the embedded HTTP server and afterward transformed from SOAP-XML into the proprietary XML format which is the only accessible interface of the MileGate.

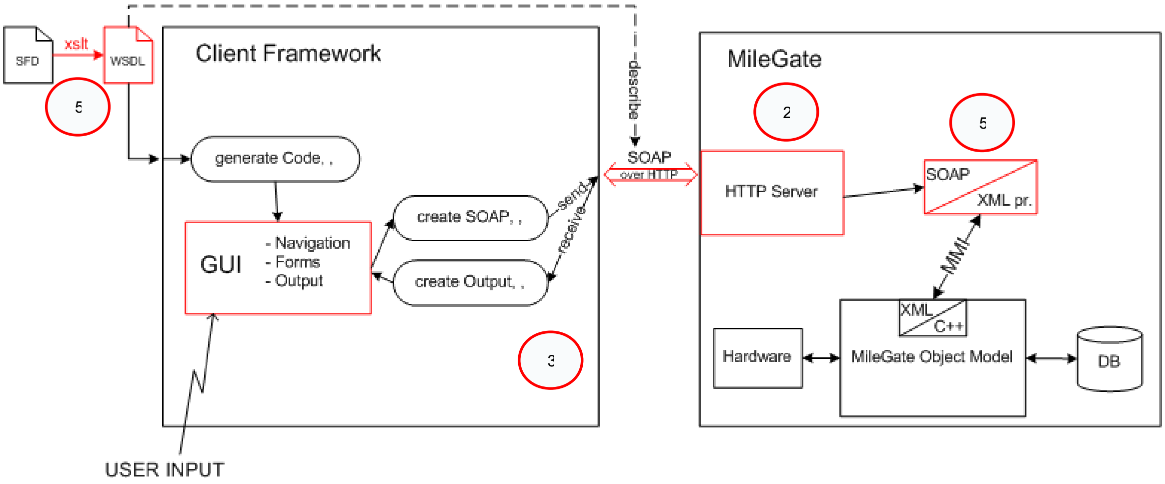


Illustration 4: Approach with web service

1.2.2.2.2 Human-Machine Interface (HMI)

To be accessible by humans, the MileGate should provide HTML files generated at runtime. Therefore, we connect the HTTP Server not with the Client framework but with a web browser.

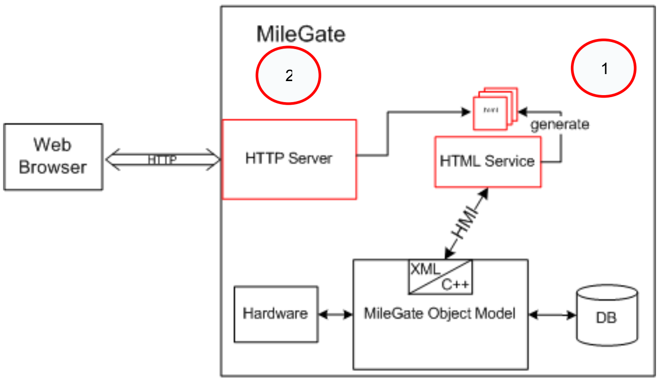


Illustration 5: Approach with generation of HTML files

For both approaches the MileGate Object Model manages the call of the C++ routines from the proprietary XML messages. In addition, it communicates with the hardware and affects storage/request of data.

1.2.3 Objectives

1.2.3.1 Side issue

Find a good way to generate on the fly HTML pages within the MilGate which providing a web browser access. 1

Survey and evaluate different embedded HTTP servers running on Linux and/or VxWorks for the MilGate. 2

1.2.3.2 Main issue

Survey and evaluate different client frameworks and describe their compatibilities with the Web Services. 3

Describe the flow of information from the KEYMILE files which describes the internal structure through the embedded HTTP Server to the MileGate 4

Define the web service and the necessary transformation. 5

Implement a prototype using the web service (MMI). 6

1.2.4 Distribution of Tasks

A strict division of the responsibilities was demanded. Even though the project was difficult to subdivide at the initiation phase, we split the tasks after some discussions as followed:

Thierry Kiki surveys and evaluates the embedded HTTP servers and client frameworks. The compatibility of the frameworks with the Web Services will be described and the client side of the prototype implemented.

David Schneider studies the feasibility for HTML generation within the MileGate and recommends a implementation. A Web Service will be described using WSDL and with it the automatic transformation to exchanged SOAP messages. The transformation from description files to WSDL and from SOAP to the proprietary XML format have to be adapted and described.
The resulting chapter are "2. Introduction into Web Services", "4. Web Service Description", "5. Web Service Concepts" and the Annexe ??

1.3 What is MileGate?

MileGate is an IP-based multi-service next-generation access platform that can support you in expanding your network so that it is fit for the future. MileGate combines carrier grade broadband access, telephony and data interface in one single, compact access platform.

By using MileGate you can migrate whole or parts of your telecommunications network to the NGN. Expand your range of services to include new, high quality Triple Play and broadband business services, and continue to provide the range of traditional telephony and data services at the same time, without having to rely on two systems.[1]

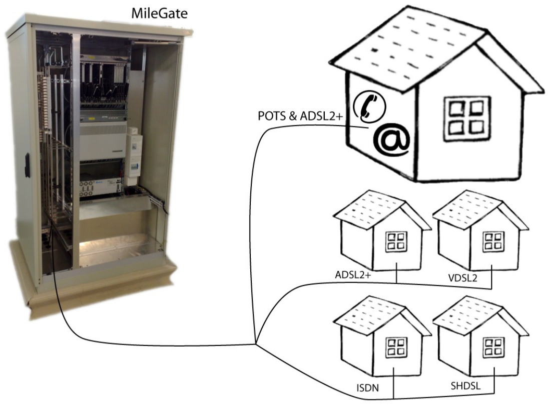


Illustration 6: MileGate

The system has one core unit and the possibility to plug 20 other units with different interfaces. As an example the MileGate provides up to 960 xDSL or 456 COMBO connections (POTS and ADSL2plus).

1.3.1 Flexibility in interfaces

Wide range of interfaces

- POTS (Plain Old Telephone Service)
- ISDN
- ADSL/ADSL2/ADSL2plus
- VDSL2
- SHDSL
- COMBO solution (POTS and ADSL2plus)
- Optical Ethernet (100BaseFx or GbE)
- Electrical Ethernet (100BaseT)
- Legacy data interfaces (E1, V.35, V.36, X.21)

[1] http://www.keymile.com/media/en/internet/products/milegate/z_brochures/MileGate_Product_Overview.pdf

1.4 General structure of the MileGate

This section describes the important parts of the MileGate structure and some mechanisms needed for the implementation of the interfaces. This section provides an abridged version of the annexe ???.

1.4.1 Structure of the Object Model

The structure has been studied on the basis of the document “Introduction to the MileGate XML Management Interface” and the actual MileGate Configuration Software Tool (MCST).
We have to clarify at the very beginning that no direct access to functions on the embedded system is provided. Information exchange with the MileGate needs to be modeled according to the Managed object model (MOM). The managed objects (MOs) are an abstract view of resources, i.e. physical or logical parts of the equipment to be managed.

The tree of Managed Objects (MO) builds a hierarchical model.

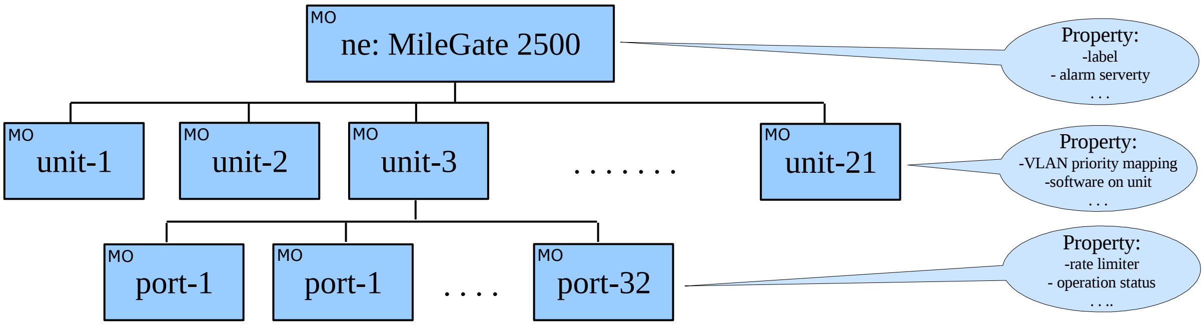


Illustration 7: MileGate Object Model structure

Each MO has its proper set of Management Functions (MF) depending on the type. The possible management functions are: Main, Configuration, Fault Management, Performance Management and Status
Each of this MF has possible properties which also depend on the type and can be changes with the MCST.

Additional information about the complete structure of the MileGate 2500 management functions, the mechanism for discovering the connected units, structure of the MileGate Accesspoint Description File (ADF), MCST GUI generation mechanism or MCST adaptaion mechanism are represented as annexe ???.

1.5 Constraints for MileGate

Due to the fact that MileGate is running embedded, there are some constraints we have to mention for the definition of our services.

1.5.1 Processor

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)

1.5.2 Memory

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)

1.6 Operating mode of MileGate

The operation mode of the MileGate was important for the definition of the interfaces as they all have to communicate with the embedded system.

1.6.1 Communication with the MileGate

For the communication with the MileGate, each management interface has another manner to communicate. As example, the communication over USB has not lot of similarities with the communication over a command line client. An very important and for all the interfaces common part is the use of KEYMILE's proprietary KOAP messages. This part of the communication is described in this section.

1.6.1.1 Client-Server system

The communication with the management interface uses a proprietary XML protocol named KOAP which is transported over a proprietary message transport protocol.

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.

1.6.1.2 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>
```

Code 8: 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>
```

Code 9: 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.

2 Introduction into Web Services

Abstract

This chapter introduces the necessary knowledge concerning the Web Service and its architecture. We also discuss the difference between traditional services and the Web Service recommended by the World Wide Web Consortium.

2.1 Traditional web service

2.2 SOA (Service-Oriented Architecture)

Before we introduce the Web Service Architecture, we need to mention some basics of the Service Oriented Architecture. This is necessary because the Web Service Architecture extends the Service Oriented Architecture.

W3C provides the following equation which interconnects the two words:

World Wide Web (WWW) + Service Oriented Architecture (SOA)
= Web Service Architecture

2.2.1 Architecture

The name indicates the basic idea behind this architecture, it is service oriented. We will not describe the SOA in detail, more information can be found under the references mentioned.

Main advantages of the SOA are that it facilitates manageable growth of enterprise systems and can reduce the costs for cooperation between organizations.

As most of the IT infrastructures and its organization have grown with a pillars-like (säulen?) architecture, the changeover to a SOA will be very difficult and time-consuming.

The following graphic illustrate this problem very well:

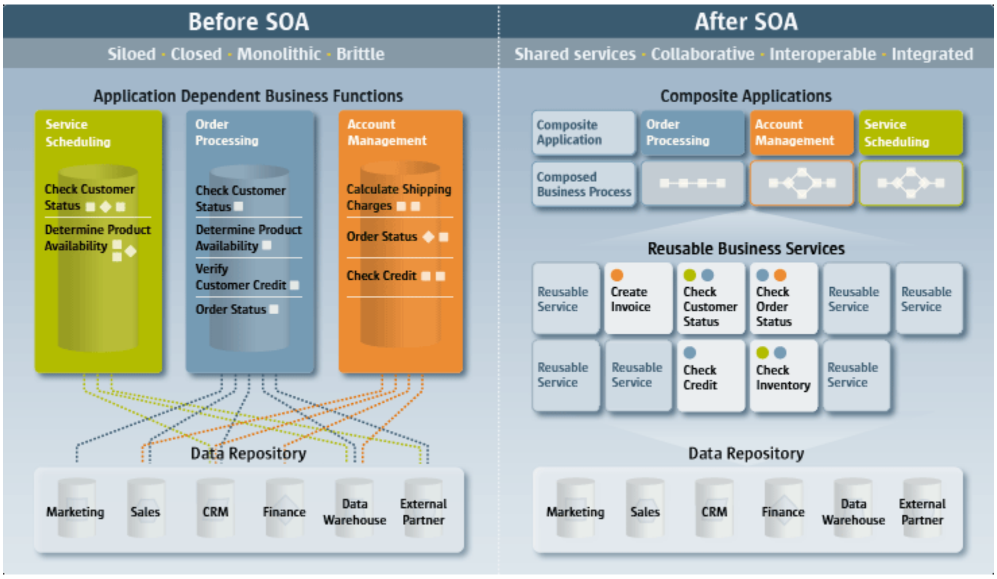


Illustration 10: Before & sfter SOA

The following illustration shows the famous triangle of Service Oriented Architectures. Roles are described briefly afterwards.

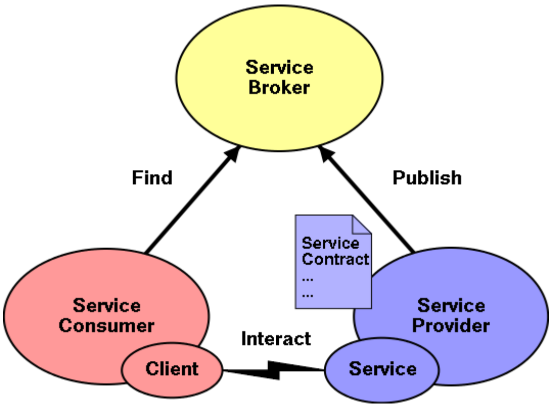


Illustration 11: Three roles in SOA

Service Provider

The service provider publishes the service. A description of the service is provided. The provider hosts and controls the access to the service.

Service Consumer

A service consumer interacts with the service via a service client. He can find services by querying the service broker. This role can be driven by an end user or by another service.

Service Broker (optional)

The service broker provides the directory service and allows service providers to publish and service costumer to find services. This role is optional, the service can also be found otherwise.

2.2.2 Basic characteristics of a SOA

A good summary of the basic characteristics of a SOA can be found in the technical library of IBM. The document is a recommendation to improve a Service Oriented Architecture and contains inter alia the following principles for a SOA.[2]

Guiding principles:

- *Reuse, granularity, modularity, composability*
- *Compliance to standards (both common and industry-specific)*
- *Services identification and categorization*

Specific architectural principles:

- *Separation of business logic from the underlying technology*
- *Single implementation and enterprise-view of components*
- *Life cycle management*
- *Efficient use of system resources*

[2] http://www.ibm.com/developerworks/webservices/library/ws-improvesoa/

2.3 Web Service Architecture

This chapter introduces the Web Service Architecture with its basic concept. We also want to introduce here the different organizations and task forces which standardize the concepts behind this architecture.
As we mentioned before, the Web Service Architecture extends a Service Oriented Architecture.

2.3.1 Definition

The definition of W3 published in the Web Services Architecture Requirements:[3]

« A Web service is a software system identified by a URI [RFC 2396], whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols. »

2.3.2 Basic Concept

The basic components of a Web Service Architecture are:

- Communication
- Service Description
- Directory Service

The W3 recommends for the communication of Web Services the use of SOAP, its specification defines the XML-based message format and how it is embedded into a transport protocol. SOAP is mostly transported over HTTP but is not at all dependent on this transport protocol.
WSDL, also XML-based, is used to describe the Web Service.
Directory service specifies a standardized structure for the management of Web Service metadata. A possible directory service is UDDI. This service, which corresponds to the Service Broker of the SOA, is optional.

[3] Source: http://www.w3.org/TR/wsa-reqs/

2.3.3 Standardization

W3C[4]:

Founded in 1994 by Tim Bernes-Lee at the Massachusetts Institute of Technology, Laboratory for Computer Science (MIT/LCS) with support of the CERN in Geneva, the DARPA (Defense Advanced Research Project Agency) and the EU (European Union).

Multiple task forces are engaged in standards for HTML, XML, SOAP and WSDL. Interesting for the future will be standards as RDF (Resource Description Framework) and OWL (Web Ontology Language) concerning the semantic web.

OASIS[5]:

The Organization for the Advancement of Structured Information Standards, originally founded in 1993 as a cooperation of commercial enterprises, has its focus on standards of the topic e-business. Beside Web Services they provide techniques as UDDI, ebXML(electronic business using XML) and WS-BPEL(Business Process Execution Language).

IETF[6]:

The Internet Engineering Task Force defines more technique oriented standards and is therefore less conspicuous on Web Service design tasks. The most important standards by IETF are TLS (Transport Layer Security), LDAP (Lightweight Directory Access Protocol) and IPv6 (Internet Protocol version 6).

WS-I[7]:

Web Service Interoperability Organization does not publish any standards. The focus lies on the examination of concrete specifications and the implementation of different producers and guarantee the interoperability of them.

Profiles were defined to describe how to use the implementation of the different producers.

[4] http://www.w3.org

[5] http://www.oasis-open.org

[6] http://www.ietf.org

[7] http://www.ws-i.org

3 Traditional web service

Abstract

This chapter continues on the traditional web service introduced before and contains our project side issues “service for HTML generation” and “survey of embedded HTTP servers”.

3.1 Embedded http server

3.2 HTML generation service

The objective of this section is according to the project objectives tofind a good way to generate on the fly HTML pages within the MilGate which is providing a web browser access.

In this section, the necessary background will be provided, the feasibility studied and at the end a proposition for the implementation given.

This section provides an abridged version of the annexe ???

3.2.1 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.

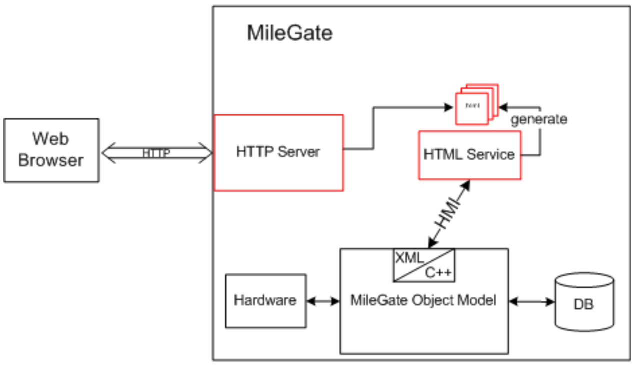


Illustration 12: System structure

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).

3.2.2 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.

3.2.3 Feasibility study

3.2.3.1 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 modifica-	If the user modifies a para-	We can not send messages

tion	meter, he needs to be sure that the operation was successful.	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.

Table 13: Problems of HTML service and mitigation

3.2.4 Recommendation for Implementation

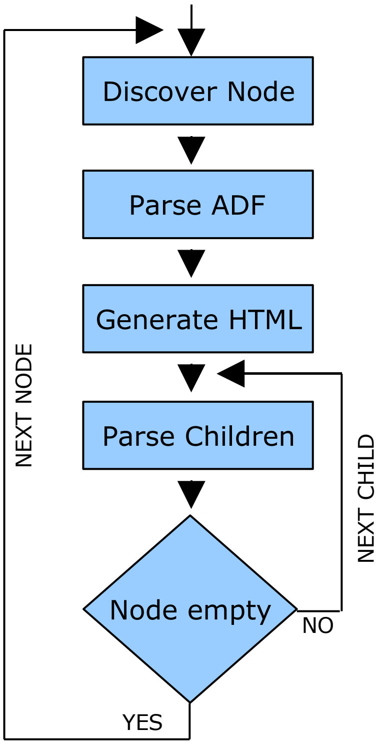
This topic contains our recommendation for the implementation of the HTML Service and an example user interface. The recommendations are based on the prior studies and converge in the basic structure towards the actual management system. This was necessary because no deep study on the structure of the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

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.

3.2.4.1 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.



Detect node (<moType>) with Discover Message

Parse the structure of the AccessPoint Definition File of each group
(mf->group->property: each property gives one page (=menu point))

The Node has to be added to the navigation menu

The Discover Message shows what kind of children <ChildrenList> are plugged to the MileGate. The childrens are parset one after another.

The children has a <state> tag which indicates if the node is empty or not.

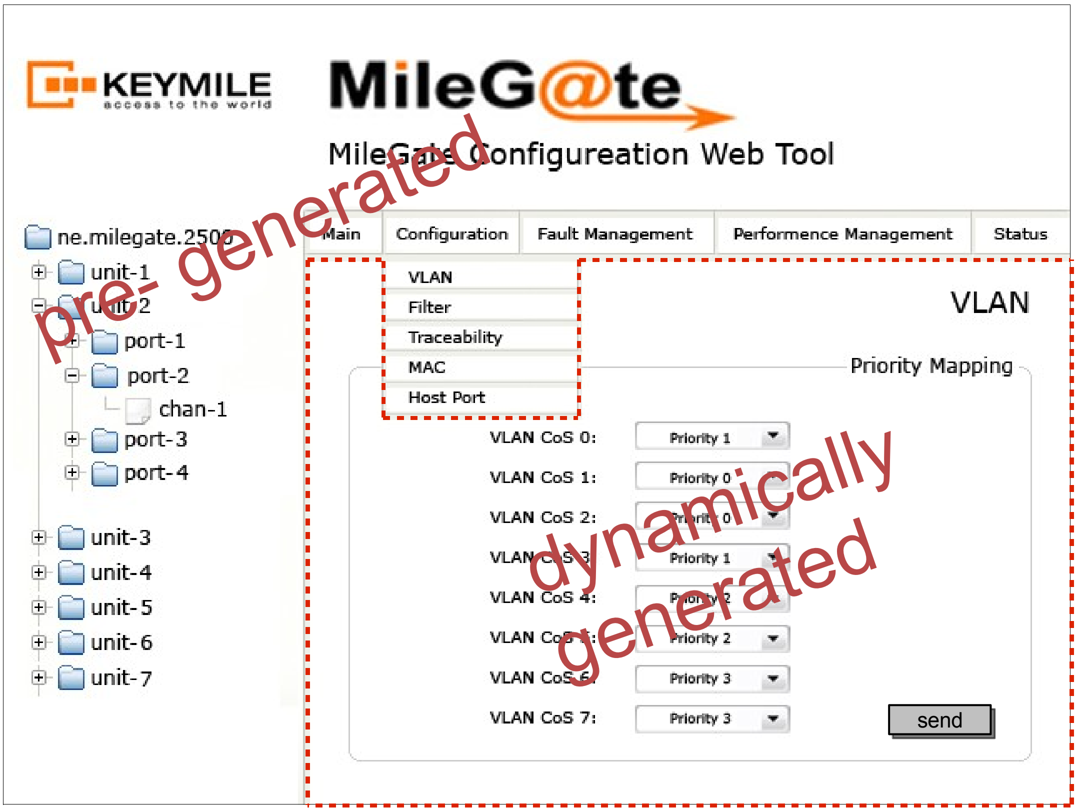
Illustration 14: Operation of the HTML Service

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

3.2.4.2 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.



VLAN CoS 0: Priority 1
VLAN CoS 1: Priority 0
VLAN CoS 2: Priority 0
VLAN CoS 3: Priority 1
VLAN CoS 4: Priority 2
VLAN CoS 5: Priority 2
VLAN CoS 6: Priority 3
VLAN CoS 7: Priority 3

Illustration 15: 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:

This list describes the activities of the program on a request of any content elements.

1. The possible form fields, check boxes, combo boxes, tables or buttons of one content frame are defined in the ADF file and need to be parsed.
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

3.2.4.3 Reaction on changes

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 as a consequence, the new navigation must be generated. This needs to be considered at the conception of the navigation structure.

If a new unit is added or removed, the node needs to be added in the navigation menu and of course also deleted. In this case, a similar mechanism as the one described in the under "Operation of the HTML service" has to be performed starting at the added unit instead of the root node.

3.2.4.4 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.

3.2.5 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 analysed 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 for configuration or supervision.

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.

4 W3C Web Service Description

Abstract

This chapter introduces the Web Service Description Language WSDL and its structure. The structure is showed by means of abstracts of the final Web Service Description for the MileGate interface.

For a better understanding, the link between the WSDL and the exchanged SOAP message has been added.

The complete description can be found as annexe.

4.1 Introduction

WSDL is an XML language for describing Web Service interfaces. The language is standardized by the W3C.

The specification of the version 1.1 exists since 2001. The follower version (2.0) reached the status of a 'W3C Recommendation' in march 2006 but most of the current Web Services still use the previous version.

The description of such a service is spit into two parties, we have an abstract and a concrete description. The abstract view focuses the functionality and the concrete enters more into the technical detail. Thus we have a separation between the details and the manner our service is offered.

The components of the description are:

Abstract:	Operation
	Messages Exchange Pattern
	Interface
Concrete:	Binding
	Endpoint
	Service

The main difference of WSDL according to other description languages for interfaces (e.g. IDL, Interface Description Language) is that everything is concentrated in one file. We are able to communicate with the service just on the base of the WSDL file. Of course we have also the possibility to write the description modular (include, import) to provide better legibility and maintainability.

4.2 Structure of the description

We want to introduce briefly the elements used to describe the Web Service and show afterwards a few more details using the description of our interface. If two elements are used to describe one single element, this is due to the different versions of WSDL. The first element belongs to the version 1.1 and the second to the standard 2.0.

definitions / description (root element)

This XML element represents the root element of the WSDL file and defines the different name spaces.

```
<?xml version="1.0" encoding="UTF-8"?>

<definitions name="mob_mainbase"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:mob_mainbase_xml="http://keymile.com/milegate/ws/mob_mainbase_xml"
   xmlns:soapbind="http://schemas.xmlsoap.org/wsdl/soap12/"
   xmlns:wsman="http://schemas.xmlsoap.org/ws/2005/06/management"
   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
   targetNamespace="http://www.keymile.com/milegate/ws/mob_mainbase_xml">
```

Code 16: WSDL definitions

documentation

The section documentation contains a textual annotation to the service.

```
   <documentation>
      -textual description of Web Service
      -further infos for the use of this service or interface
      -contact person
   </documentation>
```

Code 17: WSDL documentation

types

Defines the usable data types.

```
<types>
    <xs:schema
    xmlns="http://www.keymile.com/milegate/ws/mob_mainbase_xml"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:_0="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    targetNamespace="http://www.keymile.com/milegate/ws/mob_mainbase_xml">
       ...
       <xs:element name="Label" type="Label__Type"/>
         <xs:complexType name="Label__Type">
            <xs:sequence>
```

```
              <xs:element name="user">
                 <xs:simpleType>
                    <xs:restriction base="xs:string">
                       <xs:maxLength value="63"/>
                    </xs:restriction>
                 </xs:simpleType>
              </xs:element>
              <xs:element name="service">
                 <xs:simpleType>
                    <xs:restriction base="xs:string">
                       <xs:maxLength value="63"/>
                    </xs:restriction>
                 </xs:simpleType>
              </xs:element>
              <xs:element name="description">
                 <xs:simpleType>
                    <xs:restriction base="xs:string">
                       <xs:maxLength value="127"/>
                    </xs:restriction>
                 </xs:simpleType>
              </xs:element>
           </xs:sequence>
        </xs:complexType>
        <xs:element name="severity" type="severity__Type"/>
        <xs:simpleType name="severity__Type">
           <xs:restriction base="xs:string">
              <xs:enumeration value="notification"/>
              <xs:enumeration value="cleared"/>
              <xs:enumeration value="indeterminate"/>
              <xs:enumeration value="warning"/>
              <xs:enumeration value="minor"/>
              <xs:enumeration value="major"/>
              <xs:enumeration value="critical"/>
           </xs:restriction>
        </xs:simpleType>
        ...
```

XMLSchema base data types

Code 18: WSDL types

message

This element contains the possible messages and the types which are allowed to use.

```
 <!-- WS-Management headers  -->
  <message name="ResourceURIMessage">
     <part name="Header" element="wsman:ResourceURI"/>
  </message>
  <message name="SelectorSetMessage">
     <part name="Header"  element="wsman:SelectorSet"/>
  </message>

  <!-- WS-Addressing headers  -->
```

```
<message name="ToMessage">
   <part name="Header"  element="wsa:To"/>
</message>
<message name="ReplyToMessage">
   <part name="Header"  element="wsa:ReplyTo"/>
</message>
<message name="ActionMessage">
   <part name="Header"  element="wsa:Action"/>
</message>
<message name="MessageIDMessage">
   <part name="Header"  element="wsa:MessageID"/>
</message>

<!-- bodys  -->
<!-- FAULT MESSAGE -->
<message name="errorMessage">
   <part name="Error" element="mob_mainbase_xml:Fault"/>
</message>
...
<message name="Discover__Message">
   <part name="Body" element="mob_mainbase_xml:Discover"/>
</message>
...
   <message name="Label__Message">
   <part name="Body" element="mob_mainbase_xml:Label"/>
</message>
...
```

Reference to data type (TYPES)

Code 19: WSDL message

port type / interface

Describes the interfaces and the provided operations on this interface. For each operation the corresponding input and output messages are listed.

```
<portType name="main_base__PortType">
   <!-- MAINBASE PORT TYPES -->
   <operation name="GetLabel__Operation">
      <input message="mob_mainbase_xml:EmptyMessage"/>
      <output message="mob_mainbase_xml:Label__Message"/>
      <fault name="Fault" message="mob_mainbase_xml:errorMessage"/>
   </operation>
   <operation name="SetLabel__Operation">
      <input message="mob_mainbase_xml:Label__Message"/>
      <output message="mob_mainbase_xml:Label__Message"/>
      <fault name="Fault" message="mob_mainbase_xml:errorMessage"/>
   </operation>
   <operation name="GetAlarmSeverity__Operation">
      <input message="mob_mainbase_xml:EmptyMessage"/>
      <output message="mob_mainbase_xml:AlarmSeverity__Message"/>
      <fault name="Fault" message="mob_mainbase_xml:errorMessage"/>
   </operation>
```

The input message
(view of Service)
doesn't have any body!

```
...
<operation name="GetDiscover__Operation">
   <input message="mob_mainbase_xml:EmptyMessage"/>
   <output message="mob_mainbase_xml:Discover__Message"/>
   <fault name="Fault" message="mob_mainbase_xml:errorMessage"/>
</operation>
...
```

Code 20: WSDL portType

binding

With the element binding we declare which transport protocol is used for which interface. For inputs or outputs of operations we need to assign the messages to the elements of the transport protocol (for the example SOAP, this will be SOAP:body or SOAP:header)

```
<binding name="main_base__Interface"
     type="mob_mainbase_xml:main_base__PortType">
   <soapbind:binding style="document"
     transport="http://schemas.xmlsoap.org/soap/http"/>
   <!-- MAINBASE BINDING -->
   <operation name="GetLabel__Operation">
      <soapbind:operation
     soapAction="http://schemas.xmlsoap.org/ws/2004/09/transfer/Get"/>
      <input>
         <soapbind:header message="mob_mainbase_xml:ResourceURIMessage"
           part="Header" use="literal" />
         <soapbind:header message="mob_mainbase_xml:SelectorSetMessage"
           part="Header" use="literal" />
         <soapbind:header message="mob_mainbase_xml:ToMessage" part="Head
           er" use="literal" />
         <soapbind:header message="mob_mainbase_xml:ReplyToMessage"
           part="Header" use="literal" />
         <soapbind:header message="mob_mainbase_xml:ActionMessage"
           part="Header" use="literal" />
         <soapbind:header message="mob_mainbase_xml:MessageIDMessage"
           part="Header" use="literal" />
         <soapbind:body use="literal"/>
      </input>
      <output>
         <soapbind:header message="mob_mainbase_xml:ResourceURIMessage"
           part="Header" use="literal">
         </soapbind:header>
         <soapbind:header message="mob_mainbase_xml:SelectorSetMessage"
           part="Header" use="literal" />
         <soapbind:header message="mob_mainbase_xml:ToMessage" part="Head
           er" use="literal" />
         <soapbind:header message="mob_mainbase_xml:ReplyToMessage"
           part="Header" use="literal" />
         <soapbind:header message="mob_mainbase_xml:ActionMessage"
           part="Header" use="literal" />
         <soapbind:header message="MessageIDMessage" part="Header"
           use="literal" />
         <soapbind:body use="literal"/>
      </output>
```

Definition of SOAP transport protocol and style

Each operation has its transfer function (soapAction)

Each operation has its SOAP header and body

```
<fault name="Fault">
   <soapbind:fault use="literal" name="Fault" />
</fault>
     </operation>
     <operation name="SetLabel__Operation">
        <soapbind:operation
      soapAction="http://schemas.xmlsoap.org/ws/2004/09/transfer/Set"/>
        <input>
           <soapbind:header message="mob_mainbase_xml:ResourceURIMessage"
             part="Header" use="literal" />
           <soapbind:header message="mob_mainbase_xml:SelectorSetMessage"
             part="Header" use="literal" />
           <soapbind:header message="mob_mainbase_xml:ToMessage" part="Head
             er" use="literal" />
           <soapbind:header message="mob_mainbase_xml:ReplyToMessage"
             part="Header" use="literal" />
           <soapbind:header message="mob_mainbase_xml:ActionMessage"
             part="Header" use="literal" />
           <soapbind:header message="mob_mainbase_xml:MessageIDMessage"
             part="Header" use="literal" />
           <soapbind:body use="literal"/>
        </input>
        <output>
           <soapbind:body use="literal"/>
        </output>
        <fault name="Fault">
           <soapbind:fault use="literal" name="Fault" />
        </fault>
     </operation>
```

No need for interpretation.Passes to application as a full XML

Code 21: WSDL binding

service

Describes where the service is located. 'Services' can be subdivided into 'port/ endpoint' with different addressing parameters. See next paragraph for description of parameters.

```
  <service name="SetLabelService">
     <port name="SetLabelPort"
       binding="mob_mainbase_xml:main_base__Interface">
        <soapbind:address location="http://localhost:9357/wsman/"/>
        <wsa:EndpointReference name="labelEPR"
           xmlns:wsaw="http://www.w3.org/2006/02/addressing/wsdl">
           <wsa:Address>http://localhost:9357/wsman/</wsa:Address>
           <wsa:ReferenceParameters>
              <wsman:SelectorSet>
                 <wsman:Selector name="mf">main</wsman:Selector>
                 <wsman:Selector name="property">/Label</wsman:Selector>
              </wsman:SelectorSet>
              <wsman:ResourceURI>/unit-11</wsman:ResourceURI>
           </wsa:ReferenceParameters>
        </wsa:EndpointReference>
```

Definition of Service Endpointwith the addressing parameters

Code 22: WSDL service

4.3 SOAP Message

The SOAP message defined in the Web Service Description File helps a lot to understand the descrtiption.

We are going to represent the SOAP messages for the Set- and GetLabel operation.

GetLabel SOAP message:

```
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:mob="http://www.keymile.com/milegate/ws/mob_mainbase_xml"
    xmlns:wsa  ="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
    <soapenv:Header>
        <wsa:To>http://localhost:9357/man</wsa:To>
        <wsman:ResourceURI>/unit-11</wsman:ResourceURI>
        <wsa:ReplyTo>
            <wsa:Address>http://schemas.xmlsoap.org/ws/2004/08/addressing/
              role/anonymous</wsa:Address>
        </wsa:ReplyTo>
        <wsman:SelectorSet>
            <wsman:Selector Name="mf">main</wsman:Selector>
            <wsman:Selector Name="property">/Label</wsman:Selector>
        </wsman:SelectorSet>
        <wsa:Action>http://schemas.xmlsoap.org/ws/2004/09/transfer/Get
        </wsa:Action>
        <wsa:MessageID>urn:uuid:d2345623-bc89-4323-9e83-ueldjfued
        </wsa:MessageID>
    </soapenv:Header>
    <soapenv:Body>
    </soapenv:Body>
</soapenv:Envelope>
```

Addresse

Ressource

Property

Action: Get

Code 23: SOAP GetLabel

The input type (view of service) of the GetLabel message (defined in PortTypes):

```
<input message="mgws:EmptyMessage"/>     <!-- NO BODY -->
```

SetLabel SOAP message:

```
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:mob="http://www.keymile.com/milegate/ws/mob_mainbase_xml"
    xmlns:wsa  ="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
    <soapenv:Header>
```

```
        <wsa:To>http://localhost:9357/man</wsa:To>
        <wsman:ResourceURI>/unit-11</wsman:ResourceURI>
        <wsa:ReplyTo>
            <wsa:Address>http://schemas.xmlsoap.org/ws/2004/08/addressing/
              role/anonymous</wsa:Address>
        </wsa:ReplyTo>
        <wsman:SelectorSet>
            <wsman:Selector Name="mf">main</wsman:Selector>
            <wsman:Selector Name="property">/Label</wsman:Selector>
        </wsman:SelectorSet>
        <wsa:Action>http://schemas.xmlsoap.org/ws/2004/09/transfer/Put
        </wsa:Action>
        <wsa:MessageID>urn:uuid:d2345623-bc89-4323-9e83-ueldjfued
        </wsa:MessageID>
    </soapenv:Header>
    <soapenv:Body>
        <mob:Label>
            <mob:user>a</mob:user>
            <mob:service>b</mob:service>
            <mob:description>c</mob:description>
        </mob:Label>
    </soapenv:Body>
</soapenv:Envelope>
```

Action: Put

The SetLabel has a
body of the typ
'Label__Type'
as input message

Code 24: SOAP SetLabel

The input type (view of service) of the SetLabel message (defined in Port-Types):

```
<input message="mob_mainbase_xml:Label__Message"/>
```

5 Web Service Concepts

Abstract

There is a huge variety of concepts and standards for Web Services. Concepts are provided by the World Wide Web Consortium W3C[8], OASIS[9], Microsoft[10], IBM[11] and even more. Some of this concepts overlap.

This chapter discusses the used concepts for our Web Service and provides an selection of other concepts which has been defined during this project as interesting for the future development of the MileGate Web Service.

Most of these concepts had not been included in the actual Web Service Description (WSDL) for reasons of time constraints. For these concepts interesting points for KEYMILE are emphasized and commented.

[8] http://www.w3.org/2002/ws/

[9] http://www.oasis-open.org/specs/

[10] http://msdn.microsoft.com/en-us/library/ms951274.aspx

[11] http://www.ibm.com/developerworks/webservices/standards/

5.1 Addressing

The W3C recommendation Web Service Addressing 1.0 – Core[12] of the 9 May 2006 defines the construct of the message addressing properties and the endpoint references.

Other recommendation describes the Web Service Addressing 1.0 – SOAP Binding[13] (9 May 2006), the Web Service Addressing 1.0 – Metadata[14] (4 September 2007) and the candidate recommendation Web Service Addressing 1.0 – WSDL Binding[15] (29 May 2006).

5.1.1 WS-Addressing

This recommendation provides a mechanisms for end-to-end addressing of messages independent of the transport protocol used.

Addressing properties are, with the use of SOAP, contained in the header block.

The use of WS-Addressing allows us to address the source and destination endpoint of the system and to provide a identity for the message. Additional we specifies an action URI which defines the expected semantics.

With the concept of SOAP binding we assign the exchange structure defined by SOAP and a set of predefined faults.

The WSDL Metadata and WSDL binding indicate if the service is using WS-Addressing and provides the possibility for different message exchange patterns such as one-way, request-response, notification and solicit-response for WSDL 1.1 and some more for WSDL 2.0.

5.1.1.1 Endpoint Reference EPR

Endpoint Reference is a concept introduced by WS-Addressing and is used for the dynamic generation and customization of service endpoints.

[12] http://www.w3.org/TR/ws-addr-core/

[13] http://www.w3.org/TR/ws-addr-soap/

[14] http://www.w3.org/TR/ws-addr-metadata/

[15] http://www.w3.org/TR/ws-addr-wsdl/

As we have in our system endpoints that can change with the modification of the configuration or with the insertion of new hardware, we need a mechanism to indicate the new endpoint.
Possibilities are an additional Web Service (Endpoint Manager) which provides information about the addressable endpoints. Such a Web Service is described on the apache website (http://svn.apache.org/repos/asf/cxf/trunk/testutils/src/main/resources/wsdl/locator.wsdl).
Other approaches are described later in the chapter 'WS-Distributed Management' under 'Advertisement' and 'Discovery'.

5.1.2 WS-Management

The final specification WS-Management was published by the Distributed Management Task Force DMTF the 02 December 2008. It provides a common way for systems to access and exchange management information.

The default addressing model uses a representation of an EPR that is a tuple of the following SOAP headers:[16]

- ***wsa:To*** *(required): the transport address of the service*
- ***wsman:ResourceURI*** *(required if the default addressing model is used): the URI of the resource class representation or instance representation*
- ***wsman:SelectorSet*** *(optional): identifies or "selects" the resource instance to be accessed if more than one instance of a resource class exists*

The ResourceURI is in our case used to address the Managed Object (e.g. /unit-11) and the SelectorSet specifies the management function (mf, e.g. Main) and the property (e.g. Label).

5.1.3 WS-Transfer

WS-Management has the status of W3C Member Submission (27 September 2006). The latest working draft is dated the 25 June 2009.

We use just the defined resource operations such as get and put with the URI:

[16] http://www.dmtf.org/standards/published_documents/DSP0226_1.0.0.pdf (5.1.2 Default Addressing Model)

```
http://schemas.xmlsoap.org/ws/2004/09/transfer/Get
http://schemas.xmlsoap.org/ws/2004/09/transfer/Put
```

REMARK: In the latest working draft the URI changed to:

```
http://www.w3.org/2009/06/ws-tra/Get
http://www.w3.org/2009/06/ws-tra/Put
```

Additionally the resource operations delete and create are possible.

5.2 Resource

The concepts in this chapter describe the handling of resources with Web Services. Following specifications are published by OASIS, please pay attention on the status of the recommendation which is indicated at the beginning of each description.

5.2.1 WS-Discovery

WS-Discovery is not standardized yet and has the state of an OASIS Committee Specification 01 since 14 May 2009.[17]

It defines a discovery protocol to locate services. It is often used to discover structures like LDAP (Lightweight Directory Access Protocol) or similar directories.

As our system contains one single service per MileGate, we have no need of a discovery at this level. Discovery could be used to figure out the complete infrastructure (ensemble of MileGates). Actually, this function is not needed because the system architecture and its addressing is designed in advance and won't change over the time.

CAN'T BE USED TO DISCOVER THE MANAGED OBJECTS (RESOURCE) OF THE MILEGATE!

5.2.2 WS-Resource

WS-Resource became a OASIS Standard the 1 April 2006.

The goal of WS-Resource is to standardize the terminology and concepts needed to express the relationship between Web services and resources.[18]

A resource is represented by an endpoint reference (EPR) and addressed using the WS-Addressing concept:

[17] http://docs.oasis-open.org/ws-dd/discovery/1.1/wsdd-discovery-1.1-spec.pdf

[18] http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf (1.1 Goals and Requirements)

```
<wsa:EndpointReference>
  <wsa:Address>http://192.168.0.1?res=RessourceName</wsa:Address>
  …
</wsa:EndpointReference>
```

The SOAP binding would look as followed:

```
<wsa:To>http://192.168.0.1?res=RessourceName</wsa:To>
```

5.2.2.1 WS-Resource Properties[19]

WS-Resource Properties also became a OASIS Standard the 1 April 2006.

The goal of WS-ResourceProperties is to standardize the terminology, concepts, operations, WSDL and XML needed to express the resource properties projection, its association with the Web service interface, and the messages defining the query and update capability against the properties of a WS-Resource.

Resource Property:
A resource property is a piece of information defined as part of the state model of a WS-Resource.

Resource Properties Document:
The XML document representing a logical composition of resource property elements. The resource properties document defines a particular view or projection of the state data implemented by the WS-Resource.

5.2.2.2 Comment

This concepts offer another manner for addressing the MILEGATE property (e.g. Label) and its parameters (e.g. Label1).

- With GetMultipleResourceProperties we can get a selection of Resource Properties. This mechanism offers the possibility of a customized request according to the preferences of the client. The advantage is that we do not have to request multiple operations and filter the content afterwards.
- With QueryResourceProperties we are able to query a Resource Properties document of a WS-Resource using a query expression such as XPath.
- The manageability of the system could be improved due to the dynamic add/delete of Resource Properties into the Resource Property document.

[19] http://docs.oasis-open.org/wsrf/wsrf-ws_resource_properties-1.2-spec-os.pdf (1.1 and 2)

(InsertResourceProperties, UpdateResourceProperties, DeleteResourceProperties)

DOES NOT HELP TO FIGURE OUT WHICH ENDPOINT IS SUPPORTED BY WHICH OPERATION!

5.2.3 WS-Notification

WS-Notification contains the standard WS-Base Notification, WS-Brokered Notification and WS-Topics.

5.2.3.1 WS-Base Notification

WS-Base Notification became a OASIS Standard the 1 October 2006.

The goal of WS-BaseNotification is to standardize the terminology, concepts, operations, WSDL and XML needed to express the basic roles involved in Web services publish and subscribe for notification message exchange.[20]

A notify message containing one or more notifications should look as followed:[21]

```
…
<wsnt:Notify>
  <wsnt:NotificationMessage>
    <wsnt:SubscriptionReference>
      wsa:EndpointReferenceType
    </wsnt:SubscriptionReference> ?
    <wsnt:Topic Dialect="xsd:anyURI">
      {any} ?
    </wsnt:Topic>?
    <wsnt:ProducerReference>
      wsa:EndpointReferenceType
    </wsnt:ProducerReference> ?
    <wsnt:Message>
      {any}
    </wsnt:Message>
  </wsnt:NotificationMessage> +
    {any} *
```

[20] http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf (1.1 Goals and Requirements)

[21] http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf (3.2 Notify)

```
</wsnt:Notify>
…
```

The notify message just before is transported as content of the SOAP body. Addressing for the notification (in SOAP header) by definition is following WS-Addressing action.

```
 <wsa:Action>
   http://docs.oasis-open.org/wsn/bw-2/NotificationConsumer/Notify
</wsa:Action>
```

The concept for the management of the subscription is also defined in WS-Base Notification.

5.2.3.2 WS-Brokered Notification

WS-Topics became a OASIS Standard the 1 October 2006.

The goal of WS-BrokeredNotification is to standardize message exchanges involved in Web services publish and subscribe of a message broker.[22]

5.2.3.3 WS-Topics

WS-Topics became a OASIS Standard the 1 October 2006.

The goal of the WS-Topics specification is to define a mechanism to organize and categorize items of interest for subscription known as "topics". It defines a set of topic expression dialects that can be used as subscription expressions in subscribe request messages and other parts of the WS-Notification system.[23]

Topic:
A Topic is the concept used to categorize Notifications and their related Notification schemas.

Topic Tree:
A hierarchical grouping of Topics.

[22] http://docs.oasis-open.org/wsn/wsn-ws_brokered_notification-1.3-spec-os.pdf (1.1 Goals and Requirements)
[23] http://docs.oasis-open.org/wsn/wsn-ws_topics-1.3-spec-os.pdf (1.1 Goals and Requirements)

5.2.3.4 Comment

The mechanism described in this standards is basically similar to the notification system used in the MileGate. The requirement for the notifications used for the logbook could be fulfilled with this technique without the need for a continuous polling. (Pull-style notifications also possible)

Information in the logbook subcategories alarm, configuration changes, session login, equipment changes and events can be made accessible in a more particular way for other purposes.

It is recommended to allow authorization policies for topics.

- The hierarchical structure of the topics allows a very targeted subscription for notifications.
- Management of the topics stays handy, also for large topic sets.
- The client can regroup the readout of notification according to his belongings and anywhere in his system.

5.3 Management

We already saw the WS-Management specification in the chapter Addressing. The idea behind this separation is that we just used WS-Management for addressing purposes.

In this chapter we describe functionality that goes much further. A complex concept is represented which interconnects multiple standards described before.

5.3.1 WS-Distributed Management

The standard WS-Distributed Management contains two parties.

Management using Web Services (MUWS 1.0) became a OASIS Standard the 9 March 2005 and Management of Web Services (MOWS 1.0) on 1 August 2006. We will discuss here just the first standard. The second standard (MOWS 1.0) will be more interesting for the implementation of the management interface and not for the definition of the interface.

5.3.1.1 Management Using Web Services

The following paragraph defines some necessary terminology defined in the MUWS specification.

Manageable resource:

A resource capable of supporting one or more standard manageability capabilities.

Capability:

A group of properties, operations, events and metadata, associated with identifiable semantics and information and exhibiting specific behaviors.

Manageability capability:

A capability associated with one or more management domains.

Manageability endpoint:

A Web service endpoint associated with and providing access to a manageable resource.

Management domain:
An area of knowledge relative to providing control over, and information about, the behavior, health, lifecycle, etc. of manageable resources.

Management Using Web Services (MUWS) enables management of distributed information technology (IT) resources using Web services. Many distributed IT resources use different management interfaces. By leveraging Web service technology, MUWS enables easier and more efficient management of IT resources.[24]

MUWS is based on number of other specifications such as WS-Addressing, Metadata, Endpoint Reference, WS-Notification, WS-Topics, WS-Discovery, WS-Resource Properties which have been introduced before.

Manageability capabilities

The following capabilities are summarized from the documents MUWS part 1[25] (Chapter 3) & 2[26] (Chapter 2 and 3) mentioned as reference. The capabilities describe how the service can be used.

Operations

The operations in the MUWS specification correspond to those used in WSDL (portType element containing operation element with a description and any relevant metadata).

Properties

The properties of a manageable resource use the mechanism defined in WS-Resource Properties and its resource properties document.

Events

Event types are defined by using 'topic' and 'message content' elements. The information in the second element is transmitted as a part of the notification message (defined by WS-Base Notification).
To support event classification, different SituationCategoryTypes (element) such as AvailabilitySituation, CapabilitySituation, ConfigurationSituation and so on were defined (full list on page 9 of MUWS part 2). The aim of this classification is that the event consumer can comprehend the situation according the ability of the event source.
For each capability, topics are defined to link the capability with the event.

Metadata

[24] http://docs.oasis-open.org/wsdm/wsdm-muws1-1.1-spec-os-01.pdf (1 Introduction)

[25] http://docs.oasis-open.org/wsdm/wsdm-muws1-1.1-spec-os-01.pdf

[26] http://docs.oasis-open.org/wsdm/wsdm-muws2-1.1-spec-os-01.pdf

We can define metadata on properties and operations. The aim of this is to provide information available in WSDL and WS-Resource Properties to a tool or management application.

With the metadata element 'ValidWhile', we are able to block the invocation of an operation if certain properties do not have certain values.

Operational Status

With the capability operational status we have can simply represent if a resource is 'Available', 'PartiallyAvailable', 'Unavailable' or 'Unknown'.

This function can be implemented using the notification on property value change provided by WS-Resource Properties.

Management-related capabilities

The function of a management-related capability is related to the management of a resource, but it is not necessarily offered directly by a manageability endpoint of a resource. For example, the capability to help a manageability consumer discover a new manageable resource can be provided by a registry instead of by a management representation of the resource. As another example, a manageable resource may provide information about relationships in which it participates.

The following capabilities are summarized from the documents MUWS 2 (Chapter 4 and 5) mentioned as reference.

Relationships

The relationship defines the association between resources and the role of each participant. Interesting point for the MileGate system is that we can define a common AccessEndpoint for the participants of a relationship. A relationship may have its own properties, operations, events, lifecycles or can provide information about the relationship.

Another good point is that with the definition of relationships we enable the discovery of Endpoint References for other resource that participates in the relationship.

Advertisement

This capability provides a mechanisms to notify the creation or destruction of manageable resources. The following four new event topics are defined by Advertisement:

- ManageabilityEndpointCreation
- ManageableResourceCreation
- ManageabilityEndpointDestruction
- ManageableResourceDestruction

On the creation of a new Endpoint, the most interesting case for the MileGate system, an associated 'CreationNotification' message (WS-Notification) delivers the new Endpoint Reference.

Discovery

The goal of discovery is to obtain the EPR of a manageability endpoint.[27]

The advertisement capability, just introduced before, provides one way to provide a discovery mechanisms via events.
Another possibility is the discovery mechanisms via relationships described in under 'Relationships'.

A last possibility, perhaps also interesting for the MileGate, is the discovery of manageable resource by invoking a query on a registry. It is recommended to use a registry of the type specified by the WS-Service Group[28] specification.
Therefore MileGate should provide such a registry.

5.3.1.2 Comment

This specification defines how the different concepts can be combined together and all the advantages from each of them can enhance the usability of the complete system. We have plenty of good mechanisms for the dissolving of the problems we get if we pass from a proprietary to a standardized solution using Web Services.
It follows a short recapitulation of the advantages.

Resource Properties

- customized requests
- query resource properties using XPath
- better manageability on changes of the resource properties

Notification/Topic

- similarity to actual notification system
- hierarchical structure of topics
- subscription

Metadata

- constraints for the invocation of operations

[27] http://docs.oasis-open.org/wsdm/wsdm-muws2-1.1-spec-os-01.pdf (5 Discovery)
[28] http://docs.oasis-open.org/wsrf/wsrf-ws_service_group-1.2-spec-os.pdf

- machine readable

Operational status

- knowledge if resource is available

Relationships

- common AccessEndpoint in relationship
- discovery of Endpoint References in relationship

Advertisement

- discovery of Endpoint References with creation notifications

5.4 Conclusion

The study of the SOA was very interesting and helped to understand the advancement of the Internet in direction of Web Services. Not all the ideas are implementable for the MileGate because we have existing constraints and what is even more important, an existing and functional system. As all the transformations towards an service oriented architecture, the process will be very time-consuming.

Web Services architectures provides some exceptional concepts which offers a mass of new possibilities. Here a careful study of the requirements and on the functionalities wanted to offer had to be performed.
Attention have to be paid on the level of complexity of the system to not set limits for the implementation on the client side but also for not defining it vague or ambiguous.

"Things should be made as simple as possible, but no simpler."
Quote Albert Einstein

The endpoints of the Web Service and its management pose some problems which could be solved with the different techniques described. The easiest way to manage them is to use the endpoint references EPR by programming on the client side. A adapted version of the 'discover' (MileGate operation) which furnishes just the required information for the Web Service would be more efficient and would additionally allow to hide the infrastructure from the client.

The actual MileGate notifications, which follows the same principles as WS-Notification, should be translated into Web Service notifications and described with meta description to make it machine-readable.

For purposes of flexibility, the direct access of the management functions (not over two parameters, e.g. main/label). It will be easier to define constraints for the invocation of operations which are related to the access address (EPR).
The further idea is that we need to ensure that just possible functions can be invoked. Possibilities therefore are the simple response with an error, the 'ValidWhile' provided by WS-A Metadata or the use of relation according to WS-Distributed Management.

6 Web Service Tools

Abstract

WHAT IS IN THIS CHAPTER?

6.1 Clients Tools

6.1.1 Interoperability common problems between web services and SOAP protocol

6.1.2 Web Services Interoperability Organization (WS►I)

6.1.3 Presentation of a few frameworks

6.1.4 Framework evaluation

6.1.5 Tests tools

7 Realization of the Prototype

Abstract

This chapter describes the functionality of our prototype and describes the different stages the information runs trough.

7.1 Flow of information

For the complete flow of information we have to remember the Illustration 4 ”Approach with web service” (from chapter 1 paragraph “Work to perform”) which shows a global view of the system. The different steps are here described briefly.
Additional information can be found under the mentioned references.

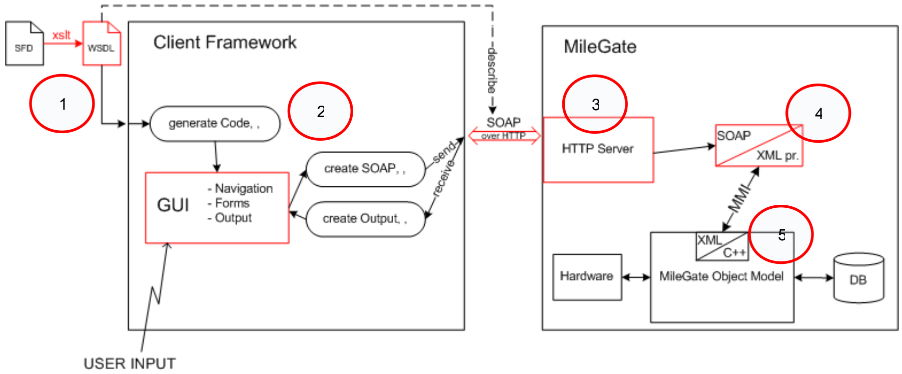


Illustration 4 ”Approach with web service” from chapter 1 paragraph “Work to perform”

7.1.1 SFD to WSDL

1 The description of the Web Service needs finally be generated automatically from files called “SFD”. Those files are provided by the system and contain almost all the necessary information for our description.

The description (WSDL file) we provided, is written manually and considers just one single interface. The complete system has more than hundred interfaces where the operations need to be accessible.

As the SFD file is written in XML, a transformation stylesheet (XSLT) will be used. The transformation from SFD to the Accesspoint Definition File ADF is already made with such a transformation file and builds the basis for the transformation to WSDL.

For this project we had to modify an existing XSLT according to the described Web Service. SOAP headers (definition and binding to message) for management and addressing had to be added. Fault type, integration into interface and fault message added and the SOAP action had to be changed.

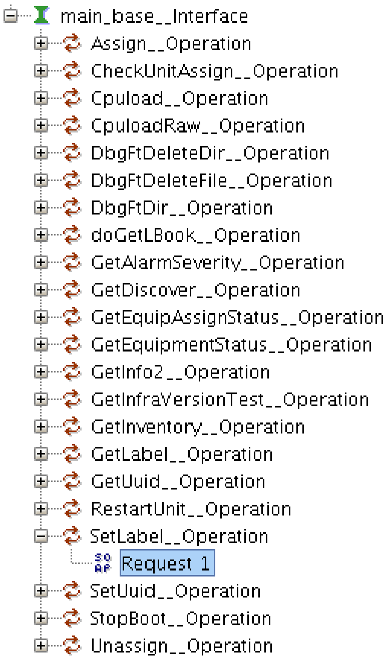
Complete XSLT provided as annex.

7.1.2 WSDL to Code & SOAP

2 The generation of the code is the job of the framework. Almost every conventional programming language provides at least one framework.
With the import of the WSDL-file, the operations defined in the description were made accessible for the programmer.

Generation of the SOAP messages is depending on the programming language but the skeleton is provided in the description.

On the right hand side we can see the automatically generated tree of the program SOAPui.
The interface (wsdl:portType) containing all the operations (wsdl:operation) defined in the WSDL file.
The SOAP request contains the defined message elements of the selected operation as header or body content.



```
main_base__Interface
    Assign__Operation
    CheckUnitAssign__Operation
    Cpuload__Operation
    CpuloadRaw__Operation
    DbgFtDeleteDir__Operation
    DbgFtDeleteFile__Operation
    DbgFtDir__Operation
    doGetLBook__Operation
    GetAlarmSeverity__Operation
    GetDiscover__Operation
    GetEquipAssignStatus__Operation
    GetEquipmentStatus__Operation
    GetInfo2__Operation
    GetInfraVersionTest__Operation
    GetInventory__Operation
    GetLabel__Operation
    GetUuid__Operation
    RestartUnit__Operation
    SetLabel__Operation
        Request 1
    SetUuid__Operation
    StopBoot__Operation
    Unassign__Operation
```

Illustration 25: SOAPui stubs

Additional information under Web Service Client.

7.1.3 HTTP Server

3 The task of the HTTP Server is to extract the payload of an HTTP request and deliver it to the system.

Baracuda sourcecode under NDA-licence.

7.1.4 SOAP to KOAP

4 We have a SOAP request arriving at the MileGate which needs to be answered by the system. The only interface to the embedded system needs KOAP requests (see Code 12: KOAP request) which have exactly the same body as our SOAP requests.
A simple DOM parser identifies the addressed unit and function and changes the syntax of the message (SOAP to KOAP and vice versa).

Source code under publishing restriction.

7.1.5 KOAP to C++

5 The invocation of the C++ routines is used for all different kinds of configuration services. Web Service can use the same manner as we changed before from SOAP to KOAP messages.
The architecture of the MileGate is hidden behind the KOAP message.

Source code under publishing restriction.

7.2 SOAP message structure

1.1

…

1.2

…

1.3

…

7.3 WSDL File generation

2.1

…

2.2

…

2.3

…

8 Tests

Abstract

This chapter contains the definition of the tests and the validation of the functionality of the prototype.

8.1 Tests Definition

This section contains the definitions for the tests we want to perform on the prototype.

8.1.1 Verification of the Web Service

The verification task for the Web Service is very important but in this case also quite difficult because the reaction of the MileGate system is predefined. We grouped the verification into two major parties. The first part is the validation which checks if the descriptions follow the standards.
The second part contains some basic tests of the system. Here we have to be aware that for lot of tests the existing software is involved which will not be modified at the moment.

8.1.1.1 Validation

For the validation of the Web Service, the most important point is that the description follows the rules defined for WDSL. With the XSLT we generate at the moment just the description for the definition WSDL 1.1. We wont validate WSDL 2.0 because the SOAP to KOAP translation in the MileGate does not support WSDL 2.0.

Also the SOAP messages need to be in accordance with the standard. This is difficult to test at the moment because the final SOAP request is generated by the client framework.

The header fields are defined according to the used standards (WS-Management and WS-Addressing) and included automatically into the SOAP message skeleton. The used Namespaces of SOAP and for the two Web Service concepts are also written automatically into the message.
For the body part of the SOAP message, the elements defined in the WSDL are included.

The validation of the Web Service functions will be performed with SOAPui 2.5.1, a Web Service Testing tool developed by eviware.
The following example shows the automatic generated SOAP message of the program SOAPui[29] (Web Service Testing Tool) of the operation 'SetLabel__operation' defined in the WSDL file. The addressing parameters are missing because the endpoint reference need to be selected by programming.

[29] http://www.soapui.org

```
<soap:Envelope
 xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
 xmlns:add="http://schemas.xmlsoap.org/ws/2004/08/addressing"
 xmlns:man="http://schemas.xmlsoap.org/ws/2005/06/management"
 xmlns:mob="http://www.keymile.com/milegate/ws/mob_mainbase_xml">
    <soap:Header>
        <add:MessageID/>
        <add:Action/>
        <add:ReplyTo/>
        <add:To/>
        <man:SelectorSet/>
        <man:ResourceURI/>
    </soap:Header>
    <soap:Body>
        <mob:Label>
            <mob:user>?</mob:user>
            <mob:service>?</mob:service>
            <mob:description>?</mob:description>
        </mob:Label>
    </soap:Body>
</soap:Envelope>
```

Code 26: SOAPui skeleton

Defined endpoint reference in the WSDL file:

```
<wsa:EndpointReference name="labelEPR"
 xmlns:wsaw="http://www.w3.org/2006/02/addressing/wsdl">
    <wsa:Address>http://localhost:9357/wsman/</wsa:Address>
    <wsa:ReferenceParameters>
        <wsman:SelectorSet>
            <wsman:Selector name="mf">main</wsman:Selector>
            <wsman:Selector name="property">/Label</wsman:Selector>
        </wsman:SelectorSet>
        <wsman:ResourceURI>/unit-11</wsman:ResourceURI>
    </wsa:ReferenceParameters>
</wsa:EndpointReference>
```

Code 27: WSDL endpoint reference

With the parameters from the endpoint reference the request looks as followed (the EPR is added in the framework):

```
<soap:Envelope
 xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
 xmlns:add="http://schemas.xmlsoap.org/ws/2004/08/addressing"
 xmlns:man="http://schemas.xmlsoap.org/ws/2005/06/management"
 xmlns:mob="http://www.keymile.com/milegate/ws/mob_mainbase_xml">
    <soap:Header>
        <add:MessageID>urn:uuid:d2345623-bc89-4323-9e83-ueldjfued</add:MessageID>
        <add:Action>http://schemas.xmlsoap.org/ws/2004/09/transfer/Get
        </add:Action>
        <add:ReplyTo>
           <add:Address>http://schemas.xmlsoap.org/ws/2004/08/addressing/role
              /anonymous</add:Address>
        </add:ReplyTo>
        <add:To>http://localhost:9357/man</add:To>
        <man:SelectorSet>                                      from EPR
           <man:Selector name="mf">main</man:Selector>
           <man:Selector name="property">/Label</man:Selector>
        </man:SelectorSet>
        <man:ResourceURI>/unit-12</man:ResourceURI>
    </soap:Header>
    <soap:Body>
        <mob:Label>
            <mob:user>?</mob:user>
            <mob:service>?</mob:service>
            <mob:description>?</mob:description>
        </mob:Label>
    </soap:Body>
</soap:Envelope>
```

Code 28: merged SOAP request

- SelectorSet and ResourceURI are specified in endpoint reference EPR
- Action is specified in the <wsdl:operation>
- MessageID and ReplyTo must be added with the framework

8.1.1.2 Testing

The testing does not completely verify if the Web Services is functioning perfectly. Testing of the function has to be verified with a framework. The aim of this part is to document the reactions on certain requests and to suggest some modifications for the actual implementation.

We want to check the reaction on malformed addressing (unit, mf, property), malformed format of the body and of course also the reaction on a well formed request. Additionally we want to check:

- Same MessageID
- No address
- Malformed ResourceURI (unit)
- Malformed Selector (mf and property)
- Malformed SOAP body

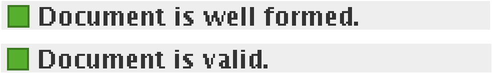
8.2 Validation of performed tests

This section contains the validation of the tests we performed on the prototype.

8.2.1 Validation of files / messages

The validation of the WSDL files was performed with the "<oXygen/> XML Editor 8.2".
The manual described WSDL file MILEGATE.wsdl and the generated (XSLT) file mob_mainbase_xml.wsdl and mob_mainequipment.wsdl had been validated for WSDL 1.1 and checked if the XML is wellformed with oXygen.

All this tree files are have the result:



🟩 Document is well formed.

🟩 Document is valid.

✓ **WSDL 1.1 validation successful**

The exchanged SOAP messages has been checked if they are wellforemed and correspond to the XML schema (http://schemas.xmlsoap.org/soap/envelope) of SOAP 1.1. This verification has been made for all the SOAP messages (request and response) described in this chapter.

✓ **SOAP 1.1 check successful**

8.2.2 Testing the response of the MileGate:

The first test is the confirmation of the functionality based on a well-formed soap request. Later we observe the reaction on malformed requests and make some suggestions.
As described in the definition of the test, the SOAP request is just partially generated by SOAPui. The endpoint references were added manually.

8.2.2.1 Well-formed request

Request (GetLabel) sent with SOAPui:

```
<!--


    GET LABEL


-->
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:mob="http://www.keymile.com/milegate/ws/mob_mainbase_xml"
    xmlns:wsa  ="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
    <soapenv:Header>
        <wsa:To>http://192.168.32.171/wsman</wsa:To>
        <wsman:ResourceURI>/unit-11</wsman:ResourceURI>
        <wsa:ReplyTo>
            <wsa:Address>http://schemas.xmlsoap.org/ws/2004/08/addressing/rol
              e/anonymous</wsa:Address>
        </wsa:ReplyTo>
        <wsman:SelectorSet>
            <wsman:Selector Name="mf">main</wsman:Selector>
            <wsman:Selector Name="proerty">www.keymile.com/mg/2008/06/MoInfo/
              Label</wsman:Selector>
        </wsman:SelectorSet>
        <wsa:Action>http://schemas.xmlsoap.org/ws/2004/09/transfer/Get
        </wsa:Action>
        <wsa:MessageID>urn:uuid:d2345623-bc89-4323-9e83-ueldjfued
        </wsa:MessageID>
    </soapenv:Header>
    <soapenv:Body/>
</soapenv:Envelope>
```

Code 29: GetLabel request

Response of MileGate:

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
    <env:Header>
        <wsa:MessageID>urn:uuid:d2345623-bc89-4323-9e83-ueldjfued-
                    resp</wsa:MessageID>
        <wsa:RelatesTo>urn:uuid:d2345623-bc89-4323-9e83-ueldj
                    fued</wsa:RelatesTo>
        <wsa:To>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonym
                    ous</wsa:To>
        <wsa:Action>http://schemas.xmlsoap.org/ws/2004/09/transfer/GetRe
                    sponse</wsa:Action>
    </env:Header>
    <env:Body>
        <Label>
            <user>?</user>
            <service>?</service>
            <description>?</description>
        </Label>
    </env:Body>
</env:Envelope>
```

Code 30: GetLabel response

Request (SetLabel) sent with SOAPui:

```
<!--



    SET LABEL



-->
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:mob="http://www.keymile.com/milegate/ws/mob_mainbase_xml"
    xmlns:wsa  ="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
    <soapenv:Header>
        <wsa:To>http://192.168.32.171/wsman</wsa:To>
        <wsman:ResourceURI>/unit-11</wsman:ResourceURI>
        <wsa:ReplyTo>
            <wsa:Address>http://schemas.xmlsoap.org/ws/2004/08/addressing/rol
              e/anonymous</wsa:Address>
        </wsa:ReplyTo>
        <wsman:SelectorSet>
            <wsman:Selector Name="mf">main</wsman:Selector>
            <wsman:Selector Name="property">
              www.keymile.com/mg/2008/06/MoInfo/Label</wsman:Selector>
        </wsman:SelectorSet>
        <wsa:Action>http://schemas.xmlsoap.org/ws/2004/09/transfer/Put
        </wsa:Action>
        <wsa:MessageID>urn:uuid:d2345623-bc89-4323-9e83-ueldjfued
        </wsa:MessageID>
    </soapenv:Header>
    <soapenv:Body>
        <mob:Label>
            <mob:user>user</mob:user>
            <mob:service>service</mob:service>
            <mob:description>description</mob:description>
        </mob:Label>
    </soapenv:Body>
</soapenv:Envelope>
```

Code 31: SetLabel request

Response of MileGate:

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
    <env:Header>
        <wsa:MessageID>urn:uuid:d2345623-bc89-4323-9e83-ueldjfued-resp
        </wsa:MessageID>
        <wsa:RelatesTo>urn:uuid:d2345623-bc89-4323-9e83-ueldjfued
        </wsa:RelatesTo>
        <wsa:To>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonym
              ous</wsa:To>
        <wsa:Action>http://schemas.xmlsoap.org/ws/2004/09/transfer/PutRe
              sponse</wsa:Action>
    </env:Header>
    <env:Body></env:Body>
</env:Envelope>
```

Code 32: SetLabel response

If we send the first request another time, we get the new values from the "SetLabel" request:

```
    ...
    <env:Body>
        <Label>
            <user>user</user>
            <service>service</service>
            <description>description</description>
        </Label>
    </env:Body>
    …
```

Code 33: new GetLabel response

Comment:
The request has some problems if the comment at the beginning of the request is removed. This error is lied to problems of the HTTP server implementation which has difficulties with the handling of too small requests. Will be solved on the next version of the HTTP server on the MileGate.

All the requests were replied successful and in a response time of between 7ms and 24ms (10 tries).

✓ **Validation successful**

8.2.2.2 Malformed request

Same MessageID

The system does not react on the message identities, it just adds the string “-resp” to the response. It is basically the job of the client programmer to ensure that the ID is unique.

A check mechanism with timer would be possible on server side which generates the predefined WS-Addressing fault wsa:DuplicateMessageID.

No address

The system does not react on a missing or mismatching address (wsa:To) in the SOAP message. This is field is actually not necessary because the address on the HTTP layer is defined with the service declaration of the interface. The principal aim of this field is to allow the forwarding of the message to another system endpoint.

If this will be implemented in the future, the match of the wsa:To and the local address should be verified. In case of success, the request should be treated, otherwise it should be forwarded or an wsa:MissingAddressInEPR returned.

Malformed ResourceURI (unit)

The system responds with the fault “Operation not found”

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
    <env:Body>
        <env:Fault>
            <env:Code>
                <env:Value>env:Sender</env:Value>
            </env:Code>
            <env:Reason>
                <env:Text xml:lang="en">
                     EXTERNAL.PLATFORM.MOB.OPERATION_NOT_FOUND</env:Text>
            </env:Reason>
        </env:Fault>
    </env:Body>
</env:Envelope>
```

Code 34: Response on malformed ResourceURI

This reaction provides the programmer the information that something with the addressing went wrong. The wanted function is not available for this resource. We suggest either to indicate the unavailability of the operation for this resource or to use the predefined wsa:InvalidEPR to keep it general.

Malformed Selector (mf and property)

The reaction of the system is for both selectors identically to the malformed ReourceURI (EXTERNAL.PLATFORM.MOB.OPERATION_NOT_FOUND).

Here it is also possible to return an wsa:InvalidEPR fault or to use something more specific as “unsupported/unknown operation”.

Malformed SOAP body

With a malformed body, means that it does not correspond the WSDL description. Actually, the service does not know the WSDL description but it has all the information provided in the ADF (accesspoint definition file) and verifies the syntax there.

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
    <env:Body>
        <env:Fault>
            <env:Code>
                <env:Value>env:Sender</env:Value>
            </env:Code>
            <env:Reason>
            <env:Text xml:lang="en">
                     EXTERNAL.PLATFORM.MOB.XML_DECODING_ERROR</env:Text>
            </env:Reason>
        </env:Fault>
    </env:Body>
</env:Envelope>
```

Code 35: Response on malformed SOAP body

The provided error description “xml decoding error” does not clearly indicate the reason and should be more precise.
We have the same reaction if we delete one element of a valid body.

Remark
Most of the suggestions will be difficult to implement because the system does not provide further information about the failure. The reaction on “same messageID” and “No address” can be responded by the service which transformates the SOAP to KOAP messages. The other reactions need provoke a error message in the base system which gives information about the internal structure away.

9 Conclusion

asdfasdf

10 Annexes

Abstract

In this last chapter you will find the references for this report and the revision history to comprehend the modifications on the document.

10.1 References

10.1.1 Keymile:

- Introduction to the MileGate XML - Management Interface
- Web Services Interface for Milegate
- User Guide – MileGate & MCST
- C++ Programming Style Guidelines, Common Part (KEYMILE Confidential)
- C++ Programming Practice Guidelines, Common Part (KEYMILE Confidential)

10.1.2 Protocol:

• WSDL 1.1,	www.w3.org/TR/wsdl
• WSDL 2.0,	www.w3.org/TR/wsdl20
• SOAP,	www.w3.org/TR/soap/
• SOAP 1.2,	www.w3.org/TR/soap12/
• XSLT 1.0,	www.w3.org/TR/xslt
• XSLT 2.0,	www.w3.org/TR/xslt20
• XHTML 1.0,	www.w3.org/TR/xhtml1/

10.1.3 Embedded Webserver

- http://www.appwebserver.org/
- http://www.goahead.com/products/webserver/Default.aspx
- http://www.koanlogic.com/klone/features.html
- http://www.allegrosoft.com/rpproduct.html
- http://barracudaserver.com/Barracuda_web_server_SDK.html
- http://www.iniche.com/webport.php

10.1.4 Service Oriented Architecture / Web Service Architecture

- W3C documents about Web Service Architecture
http://www.w3.org/2002/ws/arch/

- Reference Model for Service Oriented Architecture 1.0
http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf

- Article: What Is Service-Oriented Architecture on webservices.xml.com
http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html

- Book: Service-orientierte Architekturen mit Web Services, Konzepte – Standards – Praxis, Ingo Melzer et al., SPEKTRUM Akademischer Verlag

- Book: Web Services. Principles and Technology, Michael P. Papazoglou, PEARSON

10.1.5 Webservice description / concepts

- WS-Addressing, W3C Recommendation
http://www.w3.org/TR/ws-addr-core/

- WS-A: WSDL Binding, W3C Recommendation
http://www.w3.org/TR/ws-addr-wsdl/

- WS-A: SOAP Binding, W3C Recommendation
http://www.w3.org/TR/ws-addr-soap/

- WS-A: Metatdata, W3C Recommendation
http://www.w3.org/TR/ws-addr-metadata/

- WS-Management, Distributed Management Task Force
http://www.dmtf.org/standards/published_documents/DSP0226_1.0.0.pdf

- WS-Transfer, W3C Submission
http://www.w3.org/Submission/WS-Transfer/

- WS-Discovery, OASIS Committee Specification
http://docs.oasis-open.org/ws-dd/discovery/1.1/wsdd-discovery-1.1-spec.pdf

- WS-Base Notification, OASIS Standard

http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf

- WS-Brokered Notification, OASIS Standard
http://docs.oasis-open.org/wsn/wsn-ws_brokered_notification-1.3-spec-os.pdf

- WS-Topics, OASIS Standard
http://docs.oasis-open.org/wsn/wsn-ws_topics-1.3-spec-os.pdf

- WS-Resource, OASIS Standard
http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf

- WS-Resource Properties, OASIS Standard
http://docs.oasis-open.org/wsrf/wsrf-ws_resource_properties-1.2-spec-os.pdf

- WS-Distributed Management: Management Using Web Services MUWS Part 1, OASIS Standard
http://docs.oasis-open.org/wsdm/wsdm-muws1-1.1-spec-os-01.pdf

- WS-Distributed Management: Management Using Web Services MUWS Part 2, OASIS Standard
http://docs.oasis-open.org/wsdm/wsdm-muws2-1.1-spec-os-01.pdf

- UDDI (Universal Description, Discovery and Integration), OASIS Standard
http://uddi.org/pubs/uddi-v3.0.2-20041019.pdf

10.2 Figures

Illustration Index

10.3 Revision history

Revision			Short description of the modification	Prepared by	Approved
Doc ID	Version	Date			
TELECOM-WS	1	03/07/09	Insertion of DSCHN's documents	DSCHN	
TELECOM-WS	1.1	06/07/09	- Taskbook information - Introduction/ MileGate - Chapter introduction - New Web Service Description	DSCHN	
TELECOM-WS	1.2	07/07/09	- New chapter structure - Flow of Information	DSCHN	
TELECOM-WS	1.3	08/07/09	- Test/Verification of WS - move parts to annex		