



VAN

FP6/2004/IST/NMP/2 - 016969 VAN

Virtual Automation Networks

Work Package 8

Engineering of VAN platform for Embedded Automation Systems

Task T8.7

Engineering Tool Support for the Industrial Experimental Setups

Deliverable D08.7-1

Specification of Engineering Tool Support for the IES

Document type	: Deliverable
Document version	: Final Release
Document Preparation Date	: 30.1.2009
Classification	: Confidential
Contract Start Date	: 01.09.2005
Duration	: 31.08.2009



Project funded by the European Community
under the "Information Society Technology"
Programme (2002-2006)

Rev.	Content	Resp. Partner	Date
0.1	Document structure (draft)	Winkler, Siemens task leader	14/11/08
0.2	Changes by IfaK and Schneider	all	26/11/08
0.3	Document structure defined	all	27/11/08
0.4	Document content migration 1	all	17/12/08
0.5	Document content migration 2	all	8/1/09
0.6	Document content review 1	all	14/1/09
0.7	Document content review migration 1	all	15/1/09
0.8	Document content review migration 2	all	16/1/09
0.9	Executive Summary, Conclusion, Introduction	all	21/1/09
0.95	Pre Release WP Level	Winkler, Siemens task leader	22/1/09
0.98	Formatting and spelling correction	Winkler, Siemens task leader	29/1/09
1.0	Release Board Level	Goetz, Schneider project leader	30/1/09

Final approval	Name	Partner
Review Task Level	Hermann Winkler	Siemens
Review WP Level	Friedrich Goetz	Schneider
Review Board Level	Axel Klostermeyer	Siemens

Executive summary

This document is the deliverable D08.7-1 of the VAN project and comprises the first results from the activity of the participants within task T8.7 “Engineering Tool Support for the Industrial Experimental Setups” of work package WP8 “Specification of Engineering Tool Support for the IES” with special focus to IES support. It is a result of cooperation between the parties: CVS University Magdeburg (CVS), BUT University Brno (BUT), IfaK Magdeburg (IfaK), Schneider Electric (Schneider), Siemens AG (Siemens) and Phoenix Contact (Phoenix).

The cooperation and input of the 14 VAN partners were also coordinated by comprehensive requirement engineering technology and methods [REQUIREMENTS].

The main tasks have been the integration of a Public Key Infrastructure (PKI integration) into VAN engineering for a secure communication from the work package WP6 which is described in chapter 2.

Also the new communication technology *named based routing* has been integrated from the work package WP2 through reuse of the Java classes by a C-Library specially adapted to WP8.

To make those integrations into the already defined architecture of the VAN Stand-Alone Tool (VANSA) and the VAN Integrated Tool (VANIT), changes across all layers of the architecture have been made.

The usage of *named based routing* communication for VAN engineering (in chapter 3) affects the enhancements of the VAN engineering (chapter 5), which are used in VANSA and VANIT.

To make the enhancements and the adaptations of other work packages there was the need of a common setup for development and test purpose (chapter 4).

Also improvements and necessary enhancements have been made to the GUI and rendering engine of VANSA to be up to date with developments made by other work packages. These specifications are categorised and described starting in chapter 5.2 and also for VANIT in chapter 5.3.

In the *Conclusions* the purpose and main results of the deliverable are summarized.

The document is complemented by a *Glossary* for explanation of basic terminology and abbreviations as well as a list of *References* used in the document.

Contents

1	Introduction	7
2	PKI Integration into the Tool Concept of VAN Engineering.....	9
2.1	Introduction to PKI	9
2.2	VAN Certification Authority Structure	9
2.3	Certificates Created for VAN	12
2.4	VAN CERTIFICATES PKI ASE Class	13
2.5	Integration of the Created Certificates into the Engineering Environment	16
2.6	Certificate Bootstrapping via VAN Engineering Environment.....	18
2.7	Use of HTTPS for VAN Engineering.....	19
3	Support of Enhanced Web Service Features	21
3.1	WS-Addressing in the VAN Engineering Client.....	21
3.2	Reuse of WP2 Device Implementation.....	22
3.2.1	Architecture Overview.....	22
3.2.2	Protocol of the IPC Interface.....	24
3.2.3	Message Definition	26
3.2.4	Interface of the Library	28
3.2.5	ws_ipc_client_wrapper_export.h File Reference.....	28
4	Common Development Scenario.....	31
4.1	Description of the Common Development Scenario	31
4.2	Common Setup for All Machines	34
4.3	Setup of the Engineering Client Machine	34
4.4	Setup of the Access Point Machine.....	35
4.5	Setup of the Device Machine.....	36
4.6	Deploying the Scenario in a Virtual-Machine Based Environment.....	37
4.7	Testing the Environment.....	39
5	Enhancements of VAN Engineering Tools	46
5.1	General Enhancements.....	46
5.1.1	Supported ASEs	46
5.1.2	Reference VAN-DD File for Test Purposes and Support of the IES	47
5.2	Enhancements for VANSAs	47
5.2.1	Signing of Certificates	47
5.2.2	User Authentication.....	50
5.2.3	User Comfort.....	50
5.2.4	Support of Secure Transport Layer	52
5.2.5	Improvements of Rendering Engine	52
5.3	Enhancements for VANIT.....	55

5.3.1	STEP7.....	55
5.3.2	AutomationXplorer+	56
5.3.3	VAN Device DTM.....	59
5.3.4	VAN Communication DTM.....	60
6	Conclusion.....	62
	Glossary	63
	Terminology	63
	Abbreviations	65
	References	66

List of Figures

Figure 1:	Path through the Internet for VAN engineering	7
Figure 2:	Root CA.....	10
Figure 3:	CA Structure and Trust Relationship.....	10
Figure 4:	Certificates	12
Figure 5:	Certificate Bootstrapping.....	18
Figure 6:	HTTPS access between two VAN devices	20
Figure 7:	Asynchronous Web Service Interaction	22
Figure 8:	Overview about related components.....	24
Figure 9:	Location of the <code>ws_ipc_client_wrapper</code> library	24
Figure 10:	WS IPC - Success scenario	25
Figure 11:	WS IPC –Fault scenario.....	25
Figure 12:	Extended IPC Message for common ASEs	26
Figure 13:	WS IPC Message payload element coding	27
Figure 14:	IPC Message.....	28
Figure 15:	WP2 Sample Device and Sample Client.....	32
Figure 16:	The Common Development Scenario	33
Figure 17:	VAN sample device's user interface	36
Figure 18:	The VMware Server Login Form	37
Figure 19:	The VMware Server User Interface.....	38
Figure 20:	The Virtual Network Editor	39
Figure 21:	The "Sample Engineering Client" Main Window.....	40
Figure 22:	The Sample Engineering Client's Addressing Form	41
Figure 23:	The Sample Engineering Client's WS Operation Form.....	42
Figure 24:	The Sample Engineering Client Device Payload Configuration Form.....	43
Figure 25:	The Sample Engineering Client's Main Window After a Successful Request.....	44
Figure 26:	The VAN Sample Device User Interface.....	45

Figure 27: Certification of a VAN Device with VANSAs	49
Figure 28: Button for Certification	50
Figure 29: Login for user authentication	50
Figure 30: VANSAs Editable Window	51
Figure 31: Example of Help Functionality	52
Figure 32: The new extended class diagram of the Rendering Engine	53
Figure 33: The new extended class diagram of the TreeView	54
Figure 34: Changeable attribute projectfile	56
Figure 35: Login function of AutomationXplorer+	57
Figure 36: Changing the password of AutomationXplorer+	57
Figure 37: User Administration of AutomationXplorer+	58
Figure 38: User Properties of AutomationXplorer+	58
Figure 39: Dependencies of Documents and Components [D08.6-1]	59
Figure 40: Overview of the VAN Communication DTM	60

1 Introduction

This deliverable describes the changes for implementation by VAN engineering from all work packages. To control the given requirements for implementation, there was the need to hold all relevant requirements in one source [REQUIREMENTS]. The changes and improvements have been collected with requirement management methods and then discussed and classified into rejected and functional. The functional requirements were then confirmed in discussions and workshops and will later be indicated by their implementation progress for the next task (D08.7-2).

The key requirements were in general the requirements for migration of PKI architecture into the architecture of the VANSAs and VANITs and the additional needed ASEs for VAN engineering, e.g. for routing over many hops. Another important requirement is the integration of *named based routing*, which is implemented by WS-Addressing. During the work on task T8.7 several open questions and challenges were analysed and solutions were provided.

The communication between the VAN engineering tool and the devices is with help of PKI to achieve secured communication. The first access can be made by an encrypted https connection using the manufacturer certificate and later with an encrypted access over https using the VAN device certificate.

The goal is to enable independent VAN engineering with the help of *name based routing* between an engineering station and a VAN device. Only a standard communication mechanism must be available between two VAN devices.

E_s : Engineering Station

A_x : VAN Device

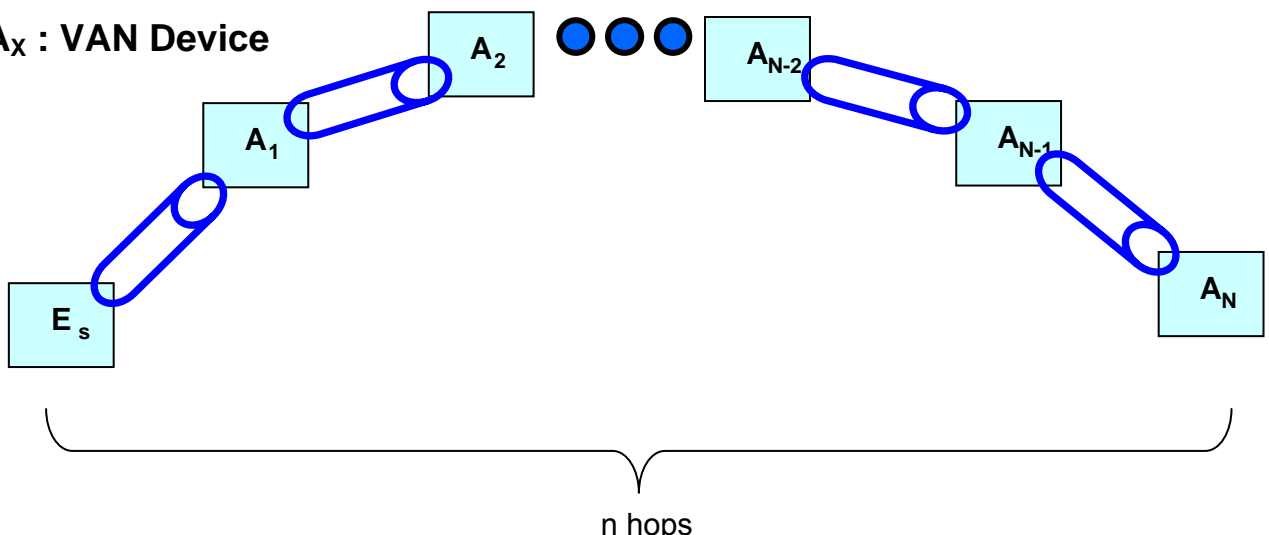


Figure 1: Path through the Internet for VAN engineering

Standard communication mechanisms enable thus independent VAN engineering with *name based routing* as described in Figure 1 from E_s to A_N .

To use this *name base routing* technology, a lot of changes described in the following chapters have been planned for the Stand-Alone and Integrated Tools. These changes have been made across the tools of VANSAs and VANITs. Thus improvements have been made in

STEP7, AutomationXplorer+, the VAN Communication DTM, the VAN Device DTM, the VAN Communication Schema Generator and the ASE schemas.

2 PKI Integration into the Tool Concept of VAN Engineering

2.1 Introduction to PKI

Concerning cryptography, **PKI** stands for public key infrastructure and is a system that connects so called **public keys** with corresponding identities (and **private keys**). The PKI is managed by a **CA (certificate authority)**.

The private key is kept secret, while the public key may be widely distributed. Private and public key are related mathematically. Important is that the private key cannot be derived from the public key.

Communication that was encrypted with the public key can only be decrypted with the corresponding private key.

A **certificate**, or more precisely a digital certificate, is an electronic document which incorporates a digital signature to bind together a public key with an identity. Certificates are **signed** by the CA.

Within a given CAs domain, each device needs only its own certificate and the **CAs public key** to authenticate every other device in that domain. A so called **CA certificate** is a digital certificate granted to one certification authority (CA) by another certification authority. In case of VAN CA certificates are needed, because of the hierarchical structure with intermediate Manufacturer and VAN CA.

In this document, all CA public keys are implemented as CA certificates. The CA certificates don't contain their corresponding private key. The CA certificates (respectively their public key part) are needed for the verification of the CAs device certificates.

2.2 VAN Certification Authority Structure

As described in [D06.4-1], VAN Engineering is the authority for the response to the initial VAN Certificate signing requests – thus the initial PKI bootstrapping of a new VAN Device. In order to fulfil this task, the VAN Engineering needs to be equipped with the correct VAN Certification Authority (CA).

Within VAN, a hierarchical CA structure is implemented:

The Root CA is hierarchically on top of the Manufacturer CA and the VAN CA. Thus, a trust relationship between the Manufacturer CA and the VAN CA can be established by checking them against the Root CA Certificate.

As described in [D06.3-1], the VAN Certification Authority and the Root CA will be stored on a portable medium like a widely spread USB-Memory Stick.

The advantage of a top-level Root CA is that new manufacturers can be introduced into VAN without the need to update the VAN CA on the USB-Memory Stick. A new manufacturer, respectively a new Manufacturer CA only has to be certified (i.e. signed) by the Root CA.

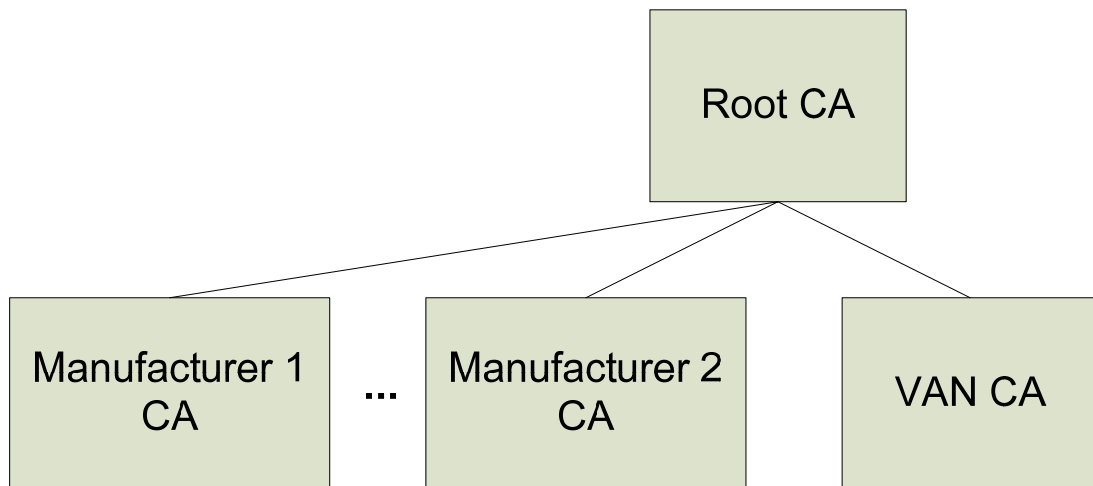


Figure 2: Root CA

This is important, because VAN implements a rather peripheral Certification Authority (CA) structure. The VAN CA is installed on an external medium i.e. a secure USB-Stick that can be attached to the VAN engineering station. Without the above structure, the update of all USB-Sticks (respectively their VAN CAs) would be the consequence.

Figure 3 shows how the different Certification Authorities collaborate within VAN. Especially important is the integration of VAN Engineering in order to allow installation and bootstrapping of new VAN Devices.

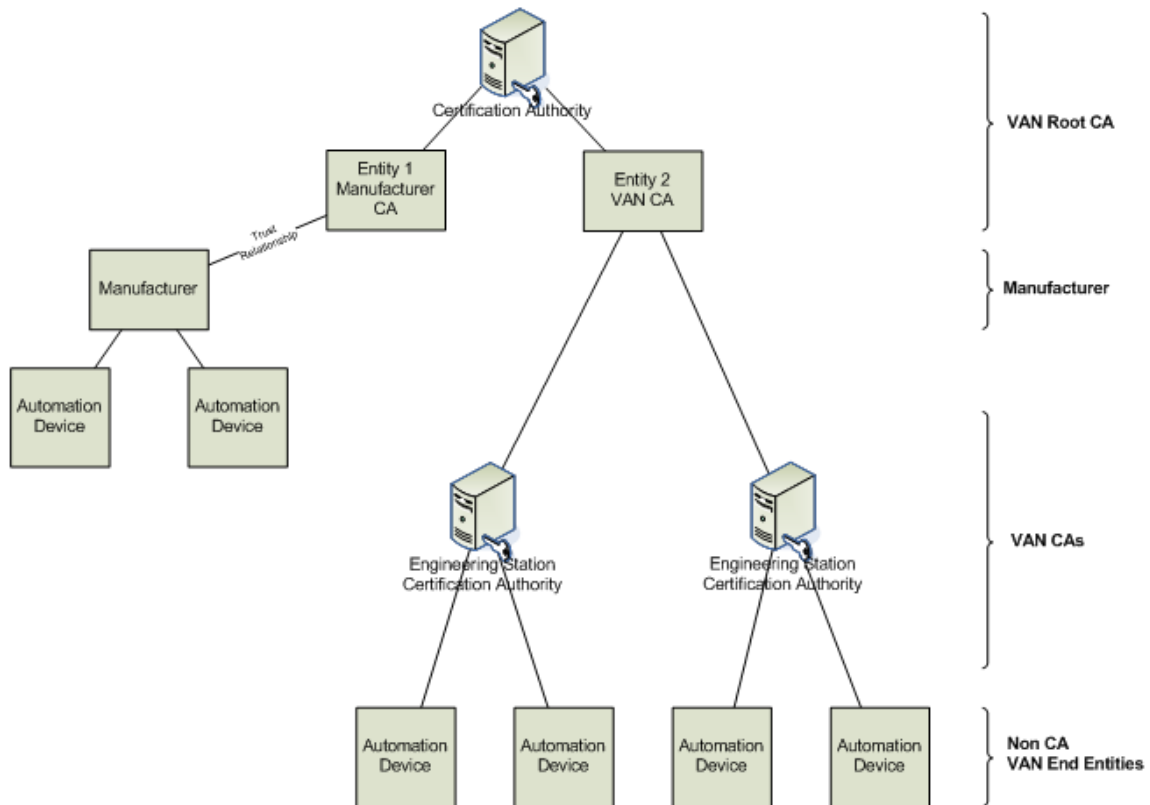


Figure 3: CA Structure and Trust Relationship

Root CA

The Root CA is the parental CA of the VAN CA and the Manufacturer CA. A copy of the Root CA Certificate is stored on the USB-Stick in order to verify the Manufacturer CAs public key during the bootstrapping process via the VAN Engineering.

Manufacturer CA

The Manufacturer CA and VAN CA are not directly related. Every manufacturer that produces VAN Devices has its own Manufacturer CA. In order to enable interoperability, trust between the Manufacturer CAs and VAN CA is established. The trust relationship was specified in D06.3-1 and D06.4-1.

Manufacturer Certificates are only used for the certificate bootstrapping process of a new VAN Device. The function of these certificates is to setup a secure initial https connection between a new VAN Device and VAN Engineering (see also Figure 5).

VAN CA

The VAN CA is used to sign the VAN Device Certificates. These VAN Device Certificates are produced and installed on the VAN devices by the device manufacturers. VAN Device Certificates are not signed after the installation. The signing of the VAN Certificates takes place after the successful connection between the VAN device and VAN Engineering took place. A copy of the VAN CA is located on the USB-Stick that is attached to the VAN Engineering.

USB-Stick

The Root and VAN Certification Authorities are not directly installed on the VAN Engineering Station, but on a portable medium. The portable medium will be a protected USB-Stick. Protection of the USB-Stick is achieved via a password (additionally a so called crypto-Stick would be possible).

The reason for the installation on a portable medium is that there is no automation server that is located in a physically secure environment available. The most appropriate possibilities to install a CA within VAN are the VAN engineering stations. But the VAN engineering stations are not located in a physically secure environment and some VAN engineering stations may also be portable. Thus the CA is not directly installed, but located on a portable medium. The CA is only used at the VAN device bootstrapping stage and then removed from the VAN engineering station.

USB-Stick Usage

The USB-Stick containing the VAN CA and the Root CA is attached to the VAN engineering station. This USB-Stick is password protected. The password has to be entered before usage.

2.3 Certificates Created for VAN

Within VAN we have two different types of certificates: VAN Device Certificates that are installed on every VAN Device for secure VAN communication and vendor specific Manufacturer Certificates that are used for the initial connection of a newly installed VAN Device and VAN Engineering.

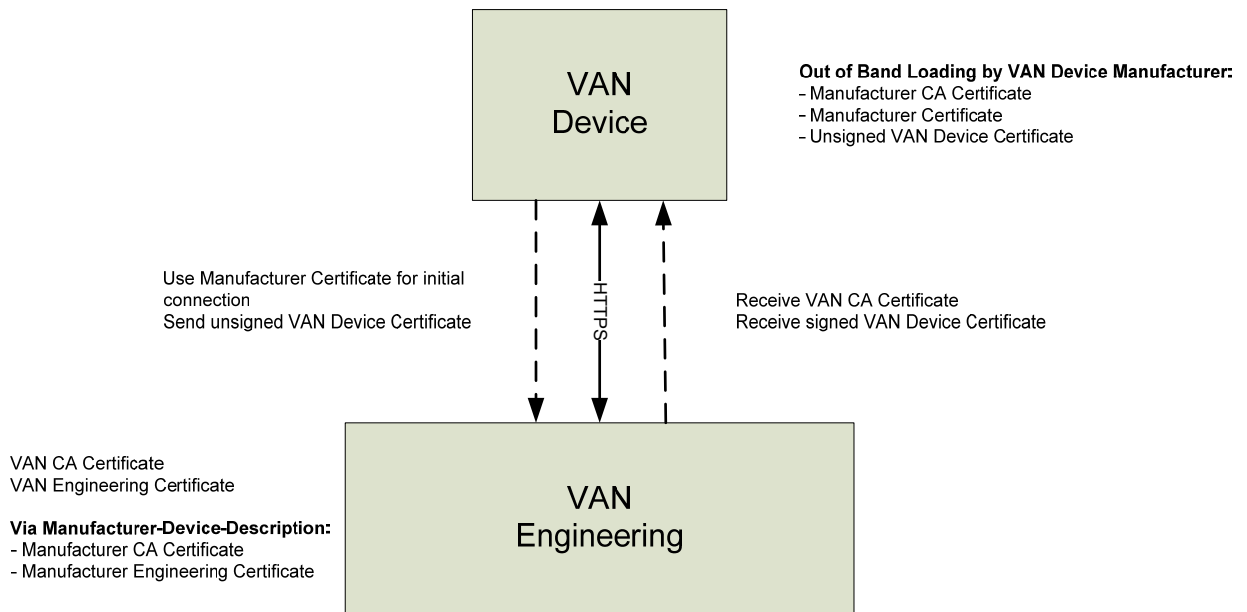


Figure 4: Certificates

Manufacturer CA Certificates

Manufacturer CA Certificates are created by the manufacturer and used to verify the signature of Manufacturer Certificates. The Manufacturer CA Certificate is a digital certificate granted to the Manufacturer CA by the Root CA. As it is only used for verification, it does not contain the corresponding private key.

Manufacturer Certificates

The Manufacturer Certificates are used to setup the initial secure https connection between the VAN Engineering and the VAN Device (see also Figure 4). Thus for initial connection the VAN Web Service is using the Manufacturer Certificate to setup https. More information about the security background for this setup can be found in deliverable D06.3-1.

- ⇒ The Manufacturer Certificate of a new VAN Device is installed on the VAN Device by the Device Manufacturer
- ⇒ The Manufacturer Engineering Certificate for VAN Engineering to connect to a new VAN Device can be found at the Manufacturer Device Description. (see also chapter 2.6).

VAN CA Certificates

A VAN CA Certificate must be installed on all VAN Devices that are going to use the VAN Device Certificate for secure communication. The VAN CA Certificate is used to verify the

public key of a VAN Device Certificate during secure communication. The VAN CA Certificate is a digital certificate granted to the VAN CA by the Root CA. As it is only used for verification, it does not contain the corresponding private key.

VAN Device Certificates

The VAN Device Certificates are not vendor specific. After successful signing of the VAN Device Certificate via VAN Engineering, the VAN Web Service is running using the VAN Device Certificate to setup https. Additionally VAN Device Certificates are used for OpenVPN connections within VAN.

More information about VAN Device Certificates can be found in [D06.3-1] and [D06.4-1].

2.4 VAN CERTIFICATES PKI ASE Class

This chapter describes how the attributes of the VAN CERTIFICATES PKI class will be used. The main purpose of the VAN CERTIFICATES PKI Class is the transfer of the unsigned VAN Device Certificate from a new VAN Device to VAN engineering and the transfer of the signed VAN Device Certificate from VAN engineering to the new VAN Device. Within the attributes of this class the relevant file-locations to fulfil this task are specified.

Concerning PKI, after fabrication, a new VAN Device contains the following folders and files:

[van_device_root]/PKI/Manufacturer

device.crt	Manufacturer Certificate
device.key	Manufacturer Certificate private key
cacert.pem	Manufacturer CA Certificate

[van_device_root]/PKI/VAN

device.csr	unsigned VAN Device Certificate
device.key	VAN Device Certificate private key

[van_device_root]/PKI/current

device.crt	Manufacturer Certificate
device.key	Manufacturer Certificate private key
cacert.pem	Manufacturer CA Certificate

VAN ASE: SECURITY CONFIG ASE

CLASS: VAN CERTIFICATES PKI

CLASS ID: 0x09A0

PARENT CLASS: COMMON SECURITY CONFIG

ATTRIBUTES:

1	(m)	Attribute:	VAN-csr
2	(m)	Attribute:	VAN-crt
3	(m)	Attribute:	VAN-pem

Restrictions

There will be only one (object) instance of this class in a VAN device.

Access to the attributes (get and set) is only allowed to the local VAN Engineering – this has to be assured by the ACL.

A getSecurityConfig is restricted to the VAN-csr attribute.

A setSecurityConfig is restricted to the VAN-crt and the VAN-pem attributes.

Unused attributes will be transferred with an empty string while using the get/set mechanism of the web service.

In case of a 'get' the value of the attribute will be directly extracted from the respective file under <van_device_root>/PKI/VAN.

In case of a 'set' the content of the received strings <VAN-crt> und <VAN-pem> will be saved as respective files under <van_device_root>/PKI/VAN.

Attributes Description

VAN-csr String

This attribute represents the content of the unsigned VAN Device Certificate file, which is stored by the manufacturer of the device at <van_device_root>/PKI/VAN.

VAN-crt String

This attribute represents the content of the signed VAN Device Certificate file, which will be stored during the very first communication with a VAN engineering tool at <van_device_root>/PKI/VAN.

VAN-pem String

This attribute represents the content of the VAN CA (Certification Authority) Certificate file, which will be stored during the very first communication with a VAN engineering tool at <van_device_root>/PKI/VAN.

The following XML-Files are needed for the transportation of the relevant Certificates during the ASE usage:

XML-File	Description / Content
setMessageResponse.xml	- Specify message response (e.g. O.K.) - Specify schema location
setMessage.xml	- Specify schema location - Set file-type and file-content
getMessageResponse.xml	- Message response (e.g. certificate signing request) - Specify schema location
certificate_pki.xml	- Specify parent class, class name and class ID - Specify schema location - Specify list of containers

getMessage.xml	<ul style="list-style-type: none">- Specify schema location- Specify get message request
----------------	---

2.5 Integration of the Created Certificates into the Engineering Environment

Integration of OpenSSL

OpenSSL has to be installed on the VAN Engineering Station in order to be able to sign the unsigned VAN Device Certificate of a new VAN Device.

The OpenSSL project has developed an open source toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security TLS (v1) protocols as well as a full-strength general purpose cryptography library.

See also: <http://www.openssl.org/>

Note: The integration will take place using OpenSSL Version 0.9.8g, because Version 0.9.8h still has problems with W32 environments and Version 0.9.8i for W32 was not yet tested.

The binary installation file of OpenSSL for W32 is provided by WP6. Alternatively it can be downloaded at:

<http://www.slproweb.com/products/Win32OpenSSL.html>

Certificates on VAN Engineering

A secure mutual connection (see also chapter 2.6) can only be established when both communicating endpoints have available (device-specific) certificates. Thus a VAN Engineering Certificate also needs to be installed on VAN Engineering.

Filename:	VAN.crt	VAN Engineering Certificate
	VAN.key	corresponding private key

File locations: [CommonAppDataFolder]/VAN/PKI/

Note: The VAN Engineering Certificate is provided by the VAN consortium and already signed. This certificate is installed on VAN Engineering during its installation.

For initial connection, the Manufacturer Certificate that is brought with the VAN Device Description is installed on VAN Engineering:

Filename:	MAN_Eng_[name].crt	Manufacturer Engineering Certificate
	MAN_Eng_[name].key	corresponding private key

File locations: [CommonAppDataFolder]/VAN/PKI/

Note: Every VAN Device Manufacturer provides his own MAN Engineering Certificate, thus [name] is a placeholder for the Manufacturer name.

Certificate Signing on VAN Engineering

As specified in deliverable D06.3-1, the preinstalled VAN Device Certificates on the VAN Devices (preinstalled by the manufacturer) are not yet signed. Thus the filename of the VAN

Device Certificate is <van_device>.csr. These Certificate Signing Requests (CSR) can only be correctly answered by the VAN CA. As specified, VAN Engineering handles the CSR.

Definition of „GET VAN Device Certificate Signing Request (CSR)“:

Use of VAN ASE: SECURITY CONFIG ASE

CLASS: VAN CERTIFICATES PKI

Attribute: unsigned-file-location

File-Location on VAN Device: [van_device_root]/PKI/VAN/VAN.csr

The command for the VAN Device Certificate signing is contained in the Batch file (csr.bat).

Definition of “Sign VAN Device Certificate”

Batch File that contains the command for VAN Certificate Signing

File location on VAN Engineering:

[CommonAppDataFolder]/VAN/PKI/csr.bat

Temporary file location of the unsigned VAN Device Certificate:

[CommonAppDataFolder]/VAN/PKI/temp/UnsignedCertificate.csr

Temporary file location of the signed VAN Device Certificate:

[CommonAppDataFolder]/VAN/PKI/temp/SignedCertificate.crt

Definition of “SET VAN Device Certificate”

Use of VAN ASE: SECURITY CONFIG ASE

CLASS: VAN CERTIFICATES PKI

Attribute: signed-file-location

File-Location on VAN Device: [van_device_root]/PKI/VAN/VAN.crt

Definition of “SET VAN CA Certificate”

Use of VAN ASE: SECURITY CONFIG ASE

CLASS: VAN CERTIFICATES PKI

Attribute: CA-file-location

File-Location on VAN Device: [van_device_root]/PKI/VAN/VAN.pem

Note: In order to enable the VAN Device Certificates to be functional on the VAN Device, the files:

VAN Device Certificate

VAN CA Certificate

VAN Device Certificate private key

have to be copied to the directory [van_device_root]/PKI/current. This is not task of VAN Engineering.

2.6 Certificate Bootstrapping via VAN Engineering Environment

The VAN Engineering environment is responsible for the VAN Device Certificate bootstrapping. As specified in D06.4-1, the VAN Device Certificate of a new VAN Device needs to be signed before its usage.

A manufacturer provides with the VAN Devices a VAN Device Description. The medium (e.g. CD-Rom) that contains the Device description additionally contains the needed certificates for the secure communication.

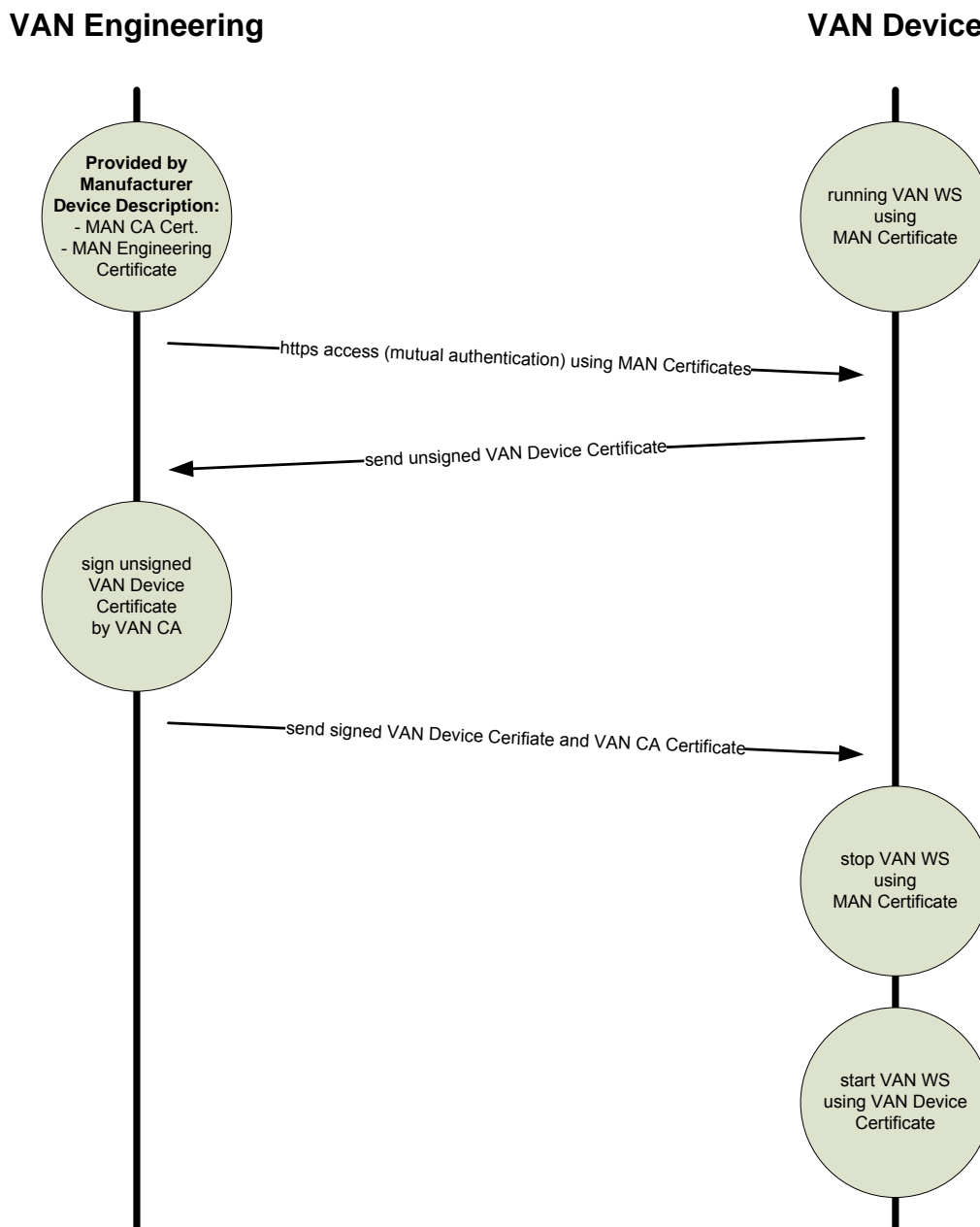


Figure 5: Certificate Bootstrapping

Mutual authentication (or sometimes just called two-way authentication) refers to two parties authenticating each other. For Figure 5, it refers to VAN Engineering authenticating to a VAN

Device and a VAN Device authenticating itself to VAN Engineering via the usage of the given certificates.

Certificate bootstrapping at VAN Engineering includes the following steps:

- ⇒ After physical connection of a new VAN Device with VAN Engineering, the Manufacturer Engineering Certificate and the Manufacturer CA Certificate are installed on VAN Engineering. This has to be done in order to enable initial https access to the new VAN Device with mutual authentication. Without the Manufacturer Engineering Certificate, VAN Engineering would not be able to make client authentication.
- ⇒ Via Manufacturer Certificate on the new VAN Device and Manufacturer Engineering Certificate on VAN Engineering a https connection (with client authentication) can be established.
- ⇒ The unsigned VAN Device Certificate is requested by the VAN Engineering via GET command using the VAN CERTIFICATES PKI Class. (see also chapter 2.4)
- ⇒ The Certificate Signing Request from the unsigned VAN Device Certificate is answered by the VAN CA using the appropriate Batch-Script on VAN Engineering (see also chapter 2.5)
- ⇒ The signed VAN Device Certificate is transferred back to the VAN Device via SET command using the VAN CERTIFICATE PKI Class.
- ⇒ After successful VAN Device Certificate bootstrapping, the VAN Device is equipped with a signed VAN Device Certificate. Thus secure connection (e.g. https) can be established using the VAN Device Certificate.

2.7 Use of HTTPS for VAN Engineering

Hypertext Transfer Protocol over Secure Socket Layer or https is a combination of the Hypertext Transfer Protocol and a network security protocol that is used to indicate a secure communication such as payment transactions and corporate information systems.

Https is not a separate protocol, but refers to the combination of a normal http interaction over an encrypted Secure Sockets Layer (SSL) or Transport Layer Security (TLS)

For a secure access through the internet, there is the need to use https also in the first access to a VAN device. For the first access, web server must be installed in the VAN device, when a manufacturer delivers the VAN device to a customer. The web server must work at an encrypted port of the web server, which is the standard https port in the internet.

Now the VAN device will be configured with ASEs to communicate with another VAN device. The VAN engineering tool configures the platform of the VAN device to make the communication available after the next reboot or by a restart of the web server.

Now the VAN engineering tool can denominate the VAN device with a certificate of the manufacturer, which interacts as reference to the secure socket layer (SSL) The activation with PKI will be made over standard https and will transport the certification to the VAN device. The VAN device has the ability to integrate the certification in his web server's configuration and will then restart itself.

When the restarting process has been finished, the https will be used with PKI encryption of the certificate further more. This means that the VAN device can only be accessed with a certificate. There is no other possible way to connect to the VAN device.

The VAN device firmware now starts the web server with the PKI encryption on an encrypted port to use openVPN for communication to the next VAN device to make the automation communication through the tunnel available. The right DNS and gateway settings for the automation connection through the tunnel must be configured using local engineering.

- HTTPS**
- Automation protocol**

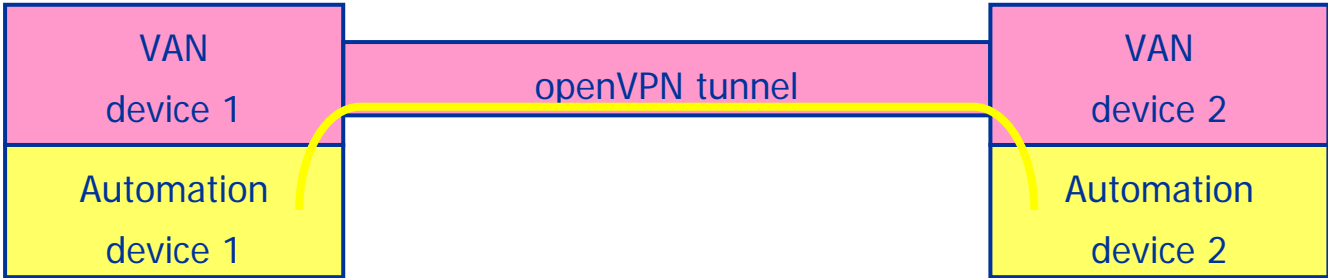


Figure 6: HTTPS access between two VAN devices

3 Support of Enhanced Web Service Features

3.1 WS-Addressing in the VAN Engineering Client

The VAN Engineering Clients are web service clients that make use of SOAP web services. After the completion of the specification, and during the implementation of the first VAN Engineering Client prototypes in tasks T8.5 and T8.6, the specification for the application of name based routing in VAN systems has been completed. To also enable the VAN Engineering Clients to access VAN devices based on their names across different domains, the WS-Addressing extension has to be integrated into the VAN Engineering Clients.

WS-Addressing is a standard which specifies how to encode routing information into SOAP messages. It is used to send SOAP messages across multiple hops regardless of the used transport protocols. Its use within the VAN project is described in depth in deliverable D02.4-1. Thus, this document focuses on the impacts of this technology on the VAN Engineering Clients.

WS-Addressing is used in the VAN project to route the messages across multiple hops, which may be found across different VAN domains. This technology makes routing independent from the underlying transport channels and allows the processing of a message to take a lot of time to transverse the routing path. Therefore, it also becomes necessary to decouple the SOAP request-response interaction's lifetime of the underlying request-response protocol. The need for this separation shortens the list of web service frameworks that can be used. Actually, only a state-of-the-art subset of the existing frameworks supports the WS-Addressing specification.

So, if there is the need to decouple the mentioned request-response lifetimes, the responses are required to be transferred through separate transport channels, so there is no risk that the channel expires while the messages are being regularly processed. This behaviour is usually called 'web services server-side asynchrony' and the web services which use it are said to be 'asynchronous web services'.

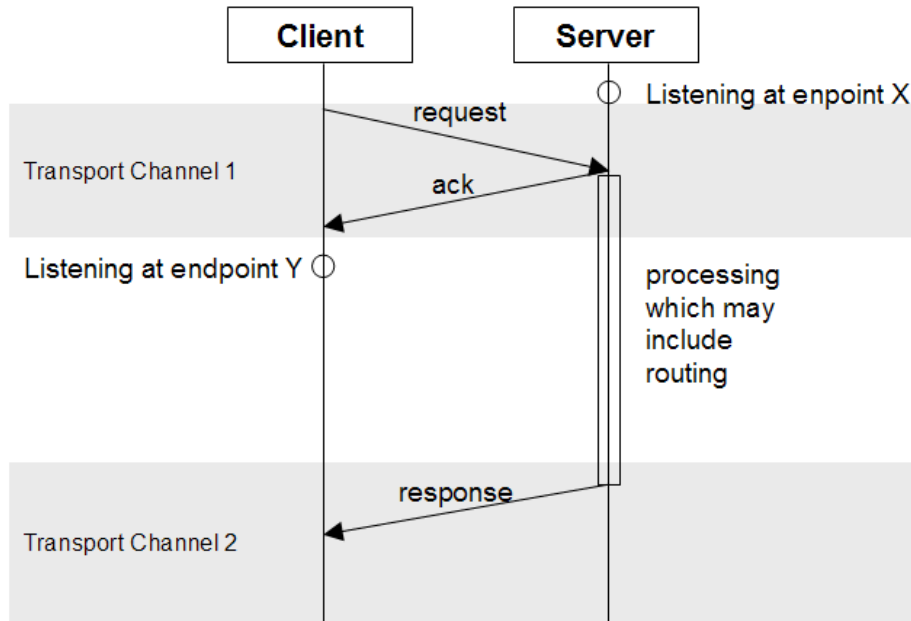


Figure 7: Asynchronous Web Service Interaction

As shown in Figure 7, an asynchronous web service implementation requires the web service caller to also behave as a server. When the server receives a request it just acknowledges the reception and later responds by using a different transport channel. Thus, the client must also listen for SOAP messages like any server would do.

As the WS-Addressing technology is quite new, many of the well-established web service frameworks currently proper support for it. This applies to the .Net framework's web service implementation, which has been used in the previous versions of the VAN Engineering Clients. As both the prototype implementations for the VAN Stand-Alone Concept and for the VAN Integrated Concept have to be modified in order to support WS-Addressing, it was decided to use a common code-base, which has partially been developed within the context of work package WP2. This new approach makes use of the state-of-the-art Axis2 framework, which is a Java-based web service framework that supports asynchronous web services. This common code-based and its reuse are described in chapter 3.2.

3.2 Reuse of WP2 Device Implementation

3.2.1 Architecture Overview

For demonstration purposes, an example implementation of a VAN device is available, offering the possibility, to support several ASE objects inside. Therefore, such a device has both communication roles, the server and the client side. The client component is used e.g. in gateway functionality. Figure 7 shows the relationship between two VAN devices. In the shown case, the remote VAN device is inactive from the communication point of view, therefore only the server role is shown. The second shown VAN device has the role of the engineering station. Therefore the client role is shown. Furthermore, the VAN Engineering Client is running on the device. From the architectural point of view it is not relevant whether the VAN Engineering Client is implemented as Stand-Alone Tool or if it is following the Integrated Tool approach.

The VAN Engineering Clients implemented in tasks T8.5 and T8.6 are not able to interact with the latest VAN devices, because there are small differences inside the available web service implementations. Therefore the VAN Engineering Client shall also use the already

available implementation of the VAN device client communication role. In order to establish a loosely coupled architecture of the software components, a specific interface called `IPC-Server` (IPC – Interprocess Communication) is established. This `IPC-Server` interface is based on TCP/IP. The resulting idea of using the available VAN device functionality is to communicate with the component, called `WS-IPC-Server` via a TCP/IP channel.

All the protocol-specific handling between the `WS-IPC-Client` and the `WS-IPC-Server`, which offer the interaction with other VAN devices is covered by a separate library `ws_ipc_client -wrapper`. This library can be very easily integrated into the Stand-Alone Tool and the Integrated Tool approach, see Figure 8. Currently the library is connected via a local TCP/IP channel to the `WS-IPC-Server`. For later usages, also a remote connection could be interesting.

The `WS-IPC-Server` listens per default at port 4711 for inbound sessions initiated by the `WS-IPC-Client`, the active communication entity. For each interaction with a VAN device, a session to `WS-IPC-Server` has to be established by means of the client, e.g. the VAN Engineering Client.

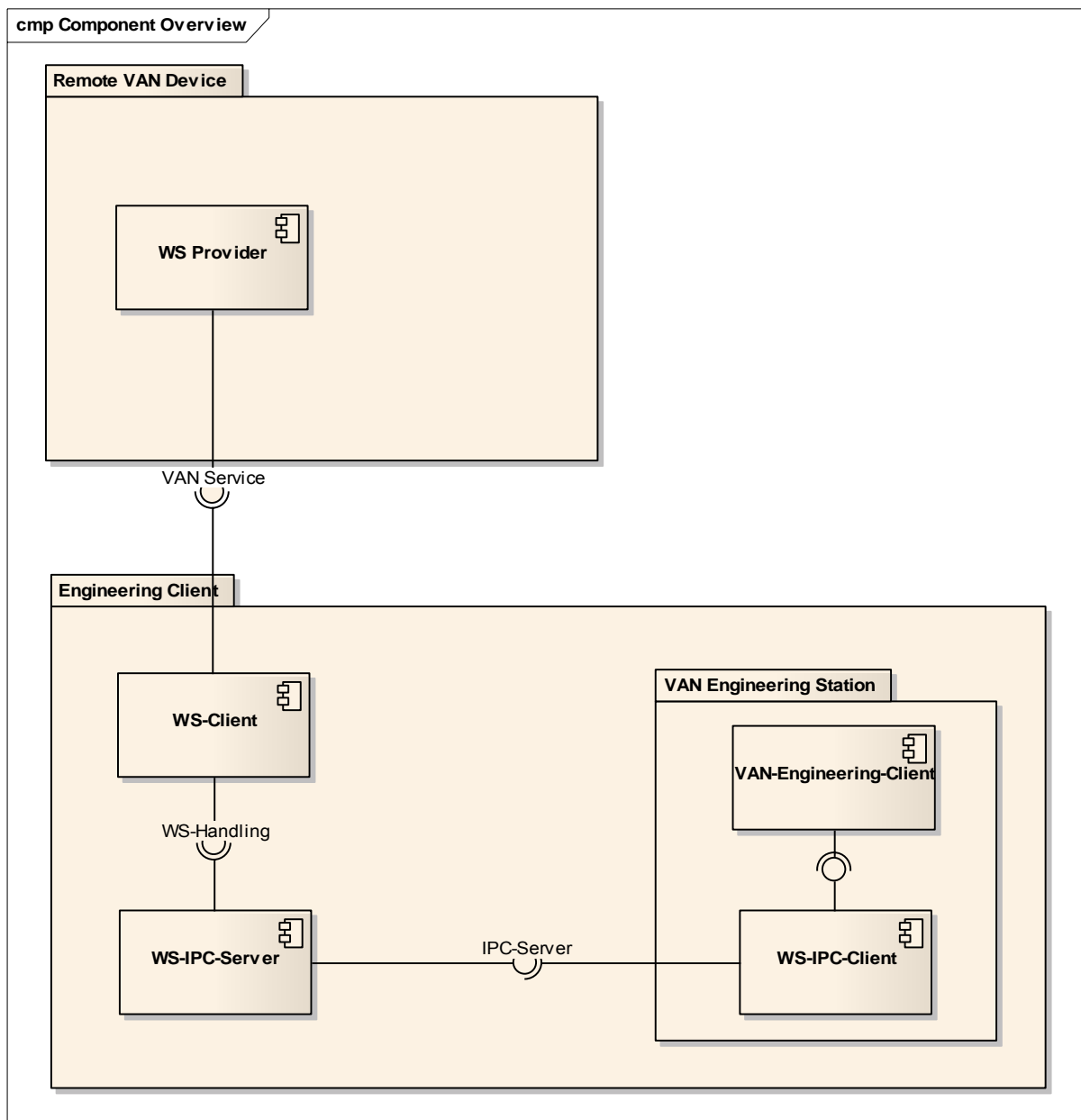
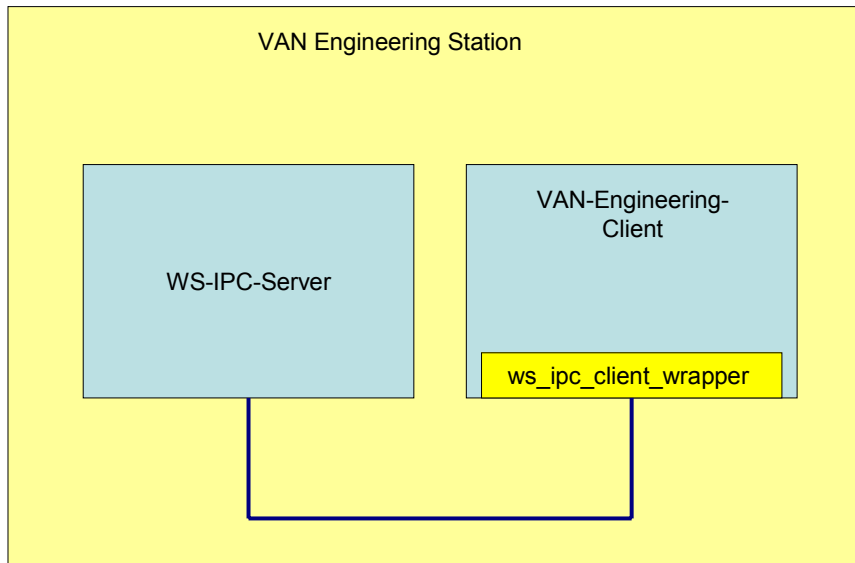


Figure 8: Overview about related components**Figure 9: Location of the `ws_ipc_client_wrapper` library**

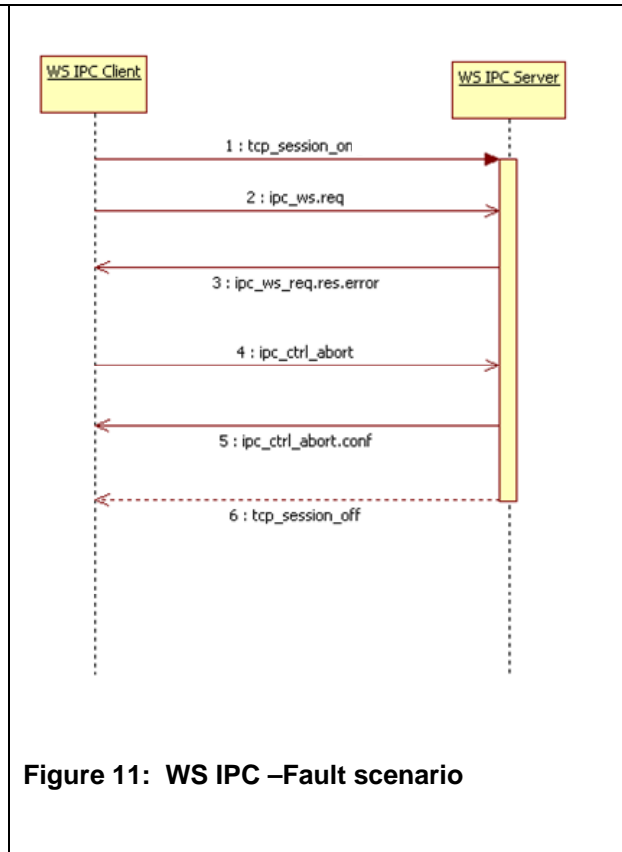
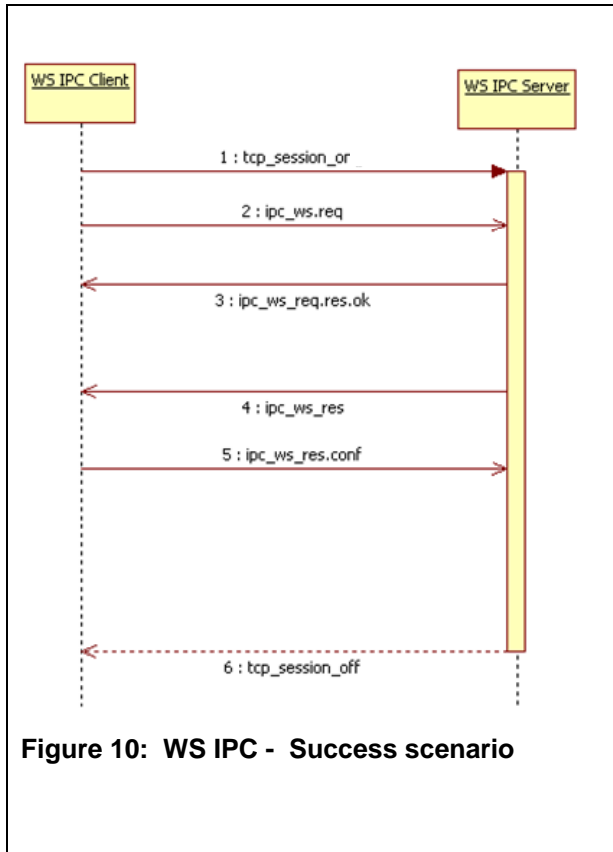
3.2.2 Protocol of the IPC Interface

The recent version of the `IPC-Server` Interface is an improvement of the specification of the VAN device implementation, which has been established for the demonstration in month 26. In contrast to the original protocol, TCP/IP is now used. This alleviates the effort when implementing the interaction between the components. Furthermore it is requested, that the client of this communication relationship establishes for each interaction with a VAN device a new session. Inside such a session, only one web service invocation (request and received response) can be handled. The complete protocol and all relevant information about the implementation of ASEs are defined in [D02.4-2].

The interactions between the `WS-IPC-Client` and `WS-IPC-Server` are shown as sequence diagrams for the two instances, see Figure 10 and Figure 11. Figure 10 shows the successful interaction and Figure 11 demonstrates the fault scenario.

Each interaction starts with the establishment of the session, shown as `tcp_session_on` and is finished by `tcp_session_off`. If the session is established, the client can transmit its request, shown as `ipc_ws.req`. If the request could be successful send the target device (internally there are several web service datagrams exchanged) and the first datagram on web service level is a successful response, this will already be signalled to the invoking client by means of `ipc_ws_req.res.ok`. Then the client has to wait further on for the response of the target device, signalled by means of `ipc_ws_res`. Depending on the success of the operation, the received data can be encoded. After this, the client commits this action by means of `ipc_ws_res.conf` and the session can be closed.

In case of errors, e.g. the target VAN device cannot be connected, a signal `ipc_ws_req.res.error` is returned from the `WS_IPC_SERVER`. Any other planned actions will be aborted, the abort is also acknowledged and the session can also be closed.



In case of errors, e.g. the target VAN device cannot be connected, a signal `ipc_ws_req.res.error` is returned from the `WS_IPC_SERVER`. Any other planned actions will be aborted, the abort is also acknowledged and the session can also be closed.

Table 1 and Table 2 describe the interaction between the components in tabular manner.

Table 1: Success scenario description

WS IPC CLIENT		WS IPC SERVER
build request	→ ws_ipc_req →	TCP socket successfully opened; IPCMessage is a Request message; First Hop to target found;
wait for ipc response	← ws_ipc_req.conf.ok ←	WS data build; WS Send out (to hop);
extract data and send res.conf.ok and release resources	← ws_ipc_res.ok ←	WS response received or
	← ws_ipc_res.fault ←	WS fault response received or
	← ws_ipc_res.timeout ←	No WS response within timeout
	→ ws_ipc_res.conf →	release resources
terminate TCP session		

Table 2: Fault scenario description

WS IPC CLIENT		WS IPC SERVER
build request	→ ws_ipc_req →	TCP socket successfully opened; IPCMessage isn't a Request message or Hop to target isn't found or isn't reachable or Stub creation error or WS send error (to hop);
Evaluate reason report to user and send ipc control abort command	← ws_ipc_req.conf.error←	Send ctrl conf and release all resources
	→ ipc_ctrl_abort →	
release resources	← ipc_ctrl_abort.conf←	
terminate TCP session		

3.2.3 Message Definition

The message format is built on top of the message format specified by the work package WP2 [D02.4-1]. The header extends the known format – that uses the fixed sized elements OperationType, OperationCode, Handle and Additional Detail by the ContentFlags element. Therefore the minimum size of a message is 20 Bytes. An IPC Message could contain message related payload content (see Figure 12).

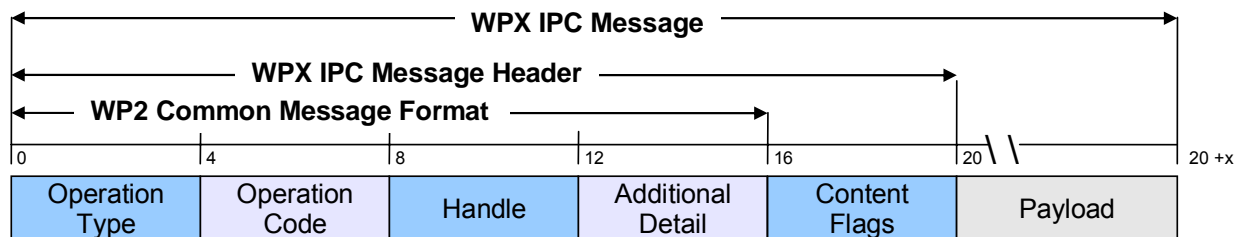


Figure 12: Extended IPC Message for common ASEs

The payload is structured in elements. A single element contains the meta-data fields that identify the type of data element and the data length followed by the data as array of bits as show in Figure 14.

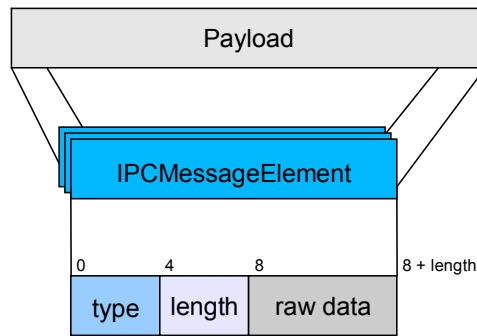


Figure 13: WS IPC Message payload element coding

The element `type` of the WS IPC-Message describes the content of the element. The types are defined as `PACKET_TYPE_XXX` information and will be inserted by means of the library as described in chapter 3.2.4. The field `Content Flags` of Figure 13 gives a summary about the content of the payload. By means of this information the server can check the received payload for completeness.

The library will put the actual parameter values of the function `vtcp_call_ws_t` (see Figure 13) into separate elements of such payload elements and will extract the response information and copy it to the given buffer.

Figure 14 gives an overview about the possible values of the different flags inside the IPC Message datagram.

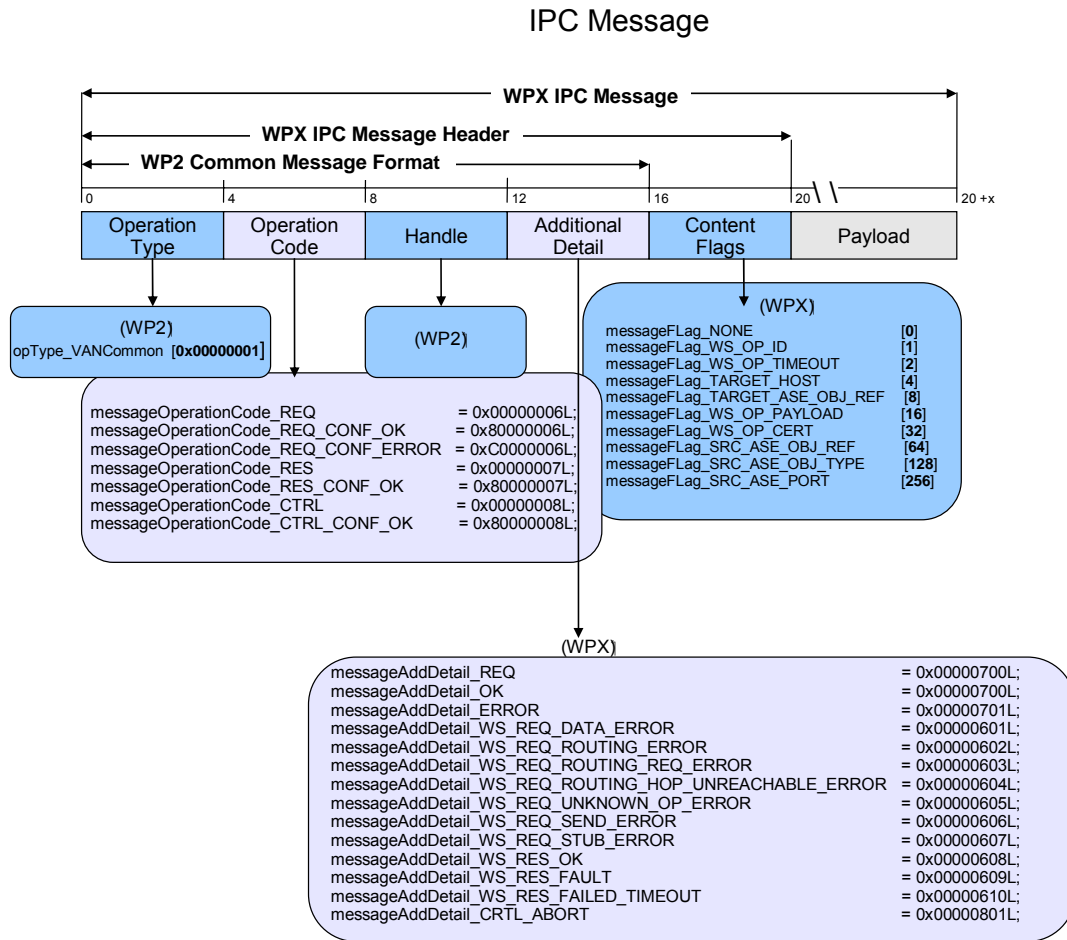


Figure 14: IPC Message

3.2.4 Interface of the Library

The library, wrapping the TCP/IP access to the VAN Engineering Client provides an interface, defined in the following subchapters. The library is developed in language C, so the library can be integrated in nearly each other implementation environment. The deployment structure of the library is a Dynamic Link Library, running on Microsoft Windows operating systems.

All information needed by the customer of the library is exported via header files. This includes all type information, coding constants etc. Especially, the offered functions are also exported via their type, meaning the signature of the functions. A signature of a function is the return type and the number and types of the formal parameters.

The customer can then very easily use the library by means of asking for the real invocation of the function addresses at runtime. So no static linking is necessary.

The following sub chapters contain the documentation of the library interface from the source code point of view. The sub chapter names are identical to the offered header files, which shall be included by the using software.

3.2.5 ws_ipc_client_wrapper_export.h File Reference

Interface for external usage of the VAN WS IPC client wrapper interface.

Data Structures

- struct **ws_ipc_server_info_s**
Information about the VAN device is running (e.g.

Define Documentation

#define VCTCP_MAJOR_VERSION_NB 1L

major version number of the interface

#define VCTCP_MINOR_VERSION_NB 0L

minor version number of the interface

Typedef Documentation

typedef struct ws_ipc_server_info_s ws_ipc_server_info_t

Information about the VAN device is running (e.g.
in a separate VM).

typedef int32_t __stdcall vctcp_get_version_t(int32_t *major, int32_t *minor)

Offers the version of the interface.

3.2.5.1.1.1 Parameters:

major - major version number, if it differs, the interface is not compatible
minor - minor version number, if it differs, only new functions were added

3.2.5.1.1.2 Returns:

VCTCP_OK - values can be used
VCTCP_NOK - wrong arguments

typedef int32_t __stdcall vctcp_set_debug_t(FILE *stream, int32_t level)

Sets the logging level according to **debug_level** definitions.

3.2.5.1.1.3 Parameters:

stream - stream, where to print the messages
level - level of logging, valid values are defined in

3.2.5.1.1.4 Returns:

VCTCP_OK - settings are accepted
VCTCP_NOK - something is wrong

typedef int32_t __stdcall vctcp_call_ws_t(const ws_ipc_server_info_t *server_info, const char *target_host, const char *target_ase_obj_ref, const u_int32_t ws_op_id, const u_int32_t timeout_in_s, const char *xml_string_tx, char *xml_string_rx, int32_t size_xml_string_rx, int32_t *error_code)

Invocation of the web service.

3.2.5.1.1.5 Parameters:

server_info - information about the WS IPC server is running (e.g. in a separate VM), normally it should run on the same machine
target_host - host name of the target VAN device
target_ase_obj_ref - instance name of the ASE for communication
ws_op_id - operation id of the web service, e.g. **OP_CODE_GetDeviceConfig**
timeout_in_s - timeout of the web service client to wait for response of the target VAN device
xml_string_tx - payload of the web service, will be passed to the ASE, content is defined in WP8

xml_string_rx - payload of the web service, will be filled by the ASE, content is defined in WP8
size_xml_string_rx - size of the provided buffer for the response of the ASE, content is defined in WP8
error_code - return code of web service client, e.g. **MSG_OP_CODE_RES_CONF_OK**

3.2.5.1.1.6Returns:

VCTCP_OK - connection to web service client was established, web service was started
VCTCP_NOK - connection to web service client could not be established

4 Common Development Scenario

The different components of the VAN Engineering Clients are developed by the partners of work package WP8 on their respective sites. This multi-site development has already shown some pitfalls in tasks T8.5 and T8.6. It was often necessary to define and setup a scenario where a specific issue could be reproduced on different sites. The additional support of WS-Addressing with the corresponding configuration of the routing information in the different devices will make it more complex to come to a common scenario. Therefore, a common development scenario will be described in this chapter and the necessary steps to setup this scenario. Moreover, the deployment of the scenario will be facilitated by providing the virtual machines which can be directly executed by corresponding virtualization software as described in chapter 4.6.

4.1 Description of the Common Development Scenario

This common development scenario has been designed in order to make it easier for the involved parties to share information about problems they may have along the improvement of the different components of the VAN Engineering Clients. The software used in this scenario is the VAN implementers' workspace, which has been developed for the work package WP2. It features a sample client and a sample VAN device with routing capabilities (Figure 15). When testing new pieces of software, some components are meant to be replaced by under-development components, like e.g. the Engineering Client shown in Figure 15.

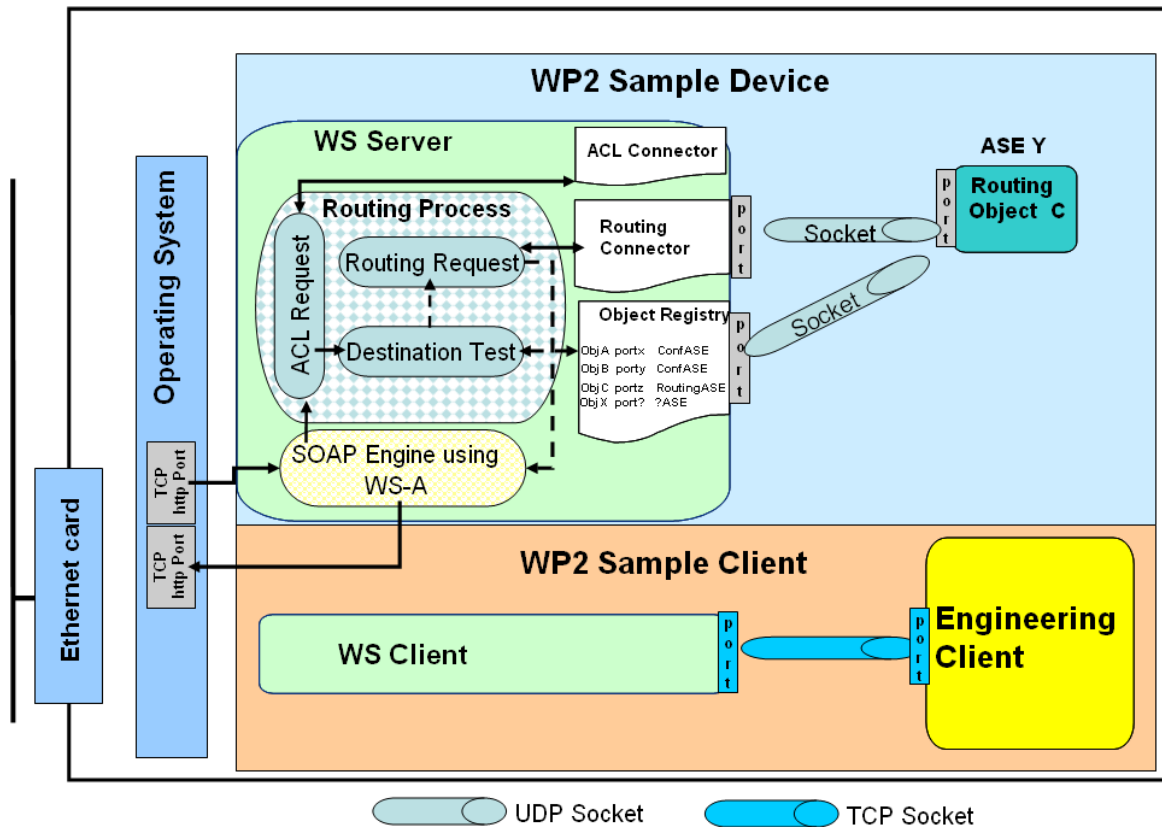


Figure 15: WP2 Sample Device and Sample Client

In order to keep the scenario simple but to include the new WS-Addressing features the developed scenario includes only one hop. Three machines take part in it: the Engineering Client Machine (ECM), the Access-Point Machine (APM), and the Device Machine (DevM).

The ECM is where the VAN Engineering Client is running. It shares its IP-subnet with the APM, which is responsible for routing the messages to the DevM in another IP-subnet. The DevM is the machine that hosts the destination device for the routed messages.

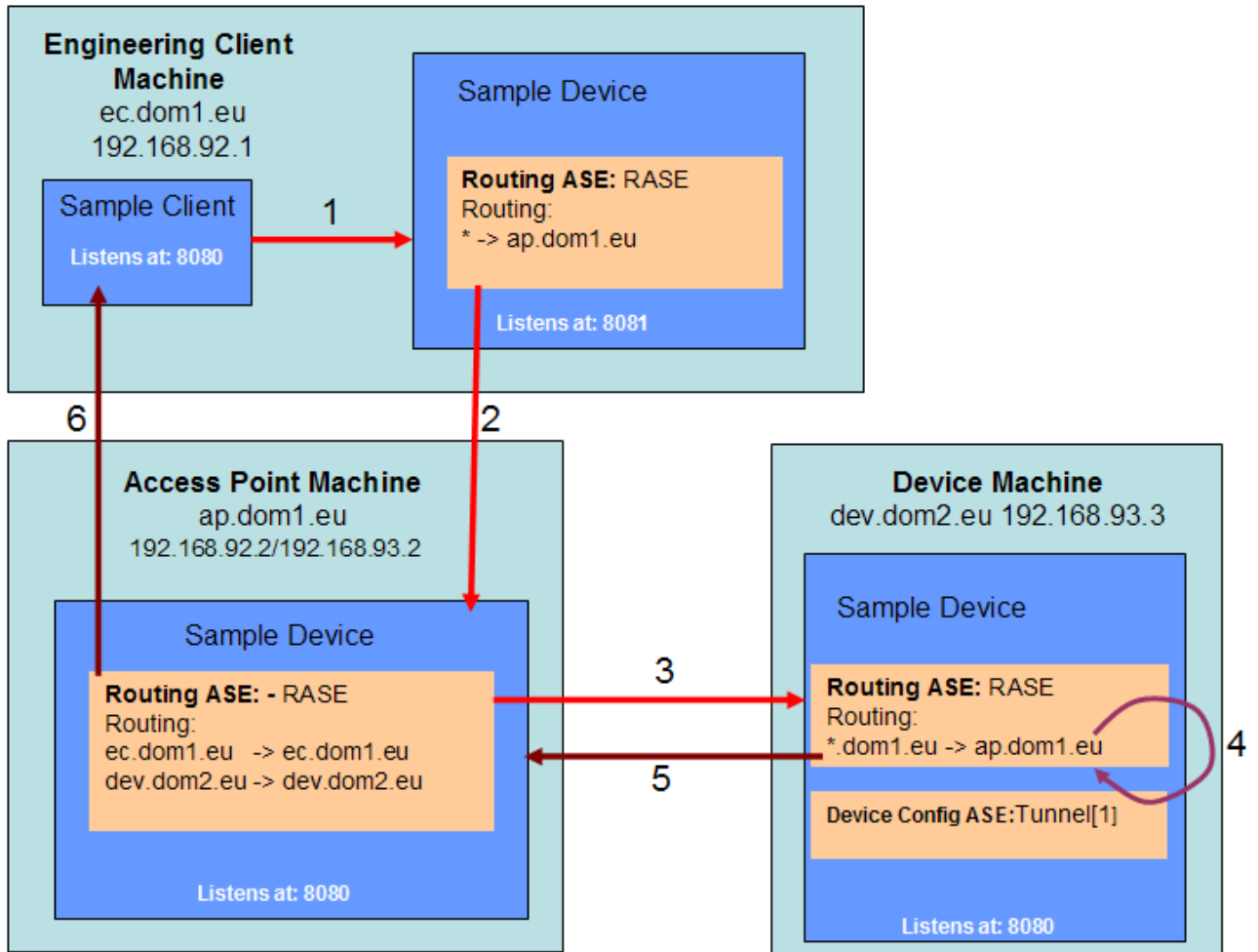


Figure 16: The Common Development Scenario

As shown in Figure 16, both the ECM and the APM have their network adapters configured to use the 192.168.92.* IP-subnet. On the other hand, the DevM's network adapter must use the 192.168.93.* IP-subnet, which is shared with the second network adapter of the APM. By these means, it is assured that the ECM cannot directly connect to the DevM and vice versa, thus guaranteeing the proper usage of WS-Addressing for the routing.

It should also be taken in account that the exchanged SOAP messages within VAN use a customized version of the WS-Addressing specification. According to the definition in deliverable D02.4-1 V2.0 the relevant fields are used as follows:

- **WSA-To** – In this field it is specified the address for the next hop to which the message is to be sent to;
- **WSA-ReplyTo** – In this field it is specified the address for the final destination of the message;
- **WSA-From** – In this field it is specified the address where the reply shall be sent to.

4.2 Common Setup for All Machines

All the machines involved in the scenario need to have the Java Software Development Kit 1.6.X installed. It can be downloaded for free at [JAVA].

The needed software modules for each machine can be found in the folder "/HOME/WP8/TASK 8.7/COMMON_DEVELOPMENT_SCENARIO" of the VAN groupware, which is subsequently mentioned as <COMMON_SCENARIO>. In this folder three zip files can be found; each one is named with the corresponding machine name (ECM, APM, DevM). Each zip file has to be extracted to an arbitrary folder of its correspondent machine. The folder for each of the machine will be subsequently mentioned as <ECM_SW>, <APM_SW> and <DEVM_SW>.

The zip files contain modified versions of the VAN implementers' workspace, which has been developed in the scope of work package WP2. These modifications are described in chapters 4.3, 4.4, and 4.5.

4.3 Setup of the Engineering Client Machine

The ECM runs both a sample client and a sample device with a Routing ASE. Thus, the sample client has only to care about the "WSA-ReplyTo" address of the sent messages, since the "WSA-To" address is always the one of the locally running sample device. This locally running sample device routes all messages according to the configuration of its Routing ASE. The sample client has to be configured to listen for asynchronous responses at port 8080 - the same port at which all VAN devices listen by default. Otherwise when routed, the responses would not be properly forwarded to the listening client port when routed. On the other hand, the locally running sample device must use port 8081, so it not only avoids a conflict with the sample client port, but also prevents the locally running sample device from taking part on the response's routing path, thus removing one unneeded hop from this path.

Before running the sample client, the machine's name resolution needs to be configured, because the common development scenario must be usable in any environment and therefore it must not use any dynamic name resolution. In order to do this, the following entries have to be added to the POSIX hosts file ('<Windows Folder>/System32/drivers/etc/hosts' on Windows):

```
192.168.92.1    ec.dom1.eu
192.168.92.2    ap.dom1.eu
```

Code Listing 1 – ECM hosts file entries

After this, the file '<COMMON_SCENARIO>/ecm.zip' has to be unzipped into an arbitrary folder on the local machine (consequently mentioned as <ECM_SW>). This contains a modified version of WP2 implementers' workspace.

The locally running device can be found in '<ECM_SW>/sample_device'. It has been modified to listen at port 8081 instead of the usual 8080, and its Routing ASE has been configured to route every received request to the APM. The way this is done is shown in Code Listing 2, which is an excerpt of the relevant information of the routing configuration file. This file can be found at '<ECM_SW>/sample_device/2507969.xml'.

```
<TARGET_DOMAIN target-subdomain-name="*">
  <list-of-next-hop>
    <next-hop next-hop-name="ap.dom1.eu"/>
  </list-of-next-hop>
</TARGET DOMAIN>
```

Code Listing 2 - Excerpt of the ECM's Routing Configuration File

The sample client has been modified in order to always send its messages to the locally running sample device, so, every SOAP message sent through this client has its "WSA-To" field set to the address of the locally running sample device.

To run the local sample device, the following command line has to be issued from the directory '<ECM_SW>/sample_device' in a command prompt:

```
> java -jar VANSampleDevice.jar
```

To run the sample client the following command line has to be issued from the directory '<ECM_SW>/sample_client' in the command prompt:

```
> java -jar VANserviceClientGui.jar
```

4.4 Setup of the Access Point Machine

The APM should have two network adapters configured for different IP-subnets. So, one of these adapters should take the IP-address '192.168.92.2' with the '255.255.255.0' subnet mask and the other second network adapter should take the IP-address '192.168.93.2' with the subnet mask '255.255.255.0'.

Before starting the sample device that is supposed to run on this machine, the name resolution and the routing configuration file for its sample device have to be configured. In order to configure the name resolution, the following entries have to be added to the POSIX hosts file ('<Windows Folder>/System32/drivers/etc/hosts' on Windows):

192.168.92.1	ec.dom1.eu
192.168.92.2	ap.dom1.eu
192.168.93.3	dev.dom1.eu

Code Listing 3 - Hosts file for the APM

After this has been done, the file '<COMMON_SCENARIO>/apm.zip' shall be extracted into an arbitrary folder on the local machine (subsequently mentioned as <APM_SW>). This contains a modified version of WP2 implementers' workspace.

The local sample device can be found in '<APM_SW>/sample_device'. Its routing table has been adequately configured to route the messages to both the ECM and the DevM. The way this has been done is shown in Code Listing 4, which is an excerpt of the relevant information of the routing configuration file. This file can be found in '<APM_SW>/sample_device/2507969.xml'.

```
<TARGET_DOMAIN target-subdomain-name="ec.dom1.eu">
  <list-of-next-hop>
    <next-hop next-hop-name="ec.dom1.eu"/>
  </list-of-next-hop>
</TARGET_DOMAIN>
<TARGET_DOMAIN target-subdomain-name="dev.dom1.eu">
  <list-of-next-hop>
    <next-hop next-hop-name="dev.dom1.eu"/>
  </list-of-next-hop>
</TARGET_DOMAIN>
```

Code Listing 4 - Excerpt of the APM's Routing Configuration File

Finally, the sample device can be run by issuing the following command line from the directory <APM_SW>/sample_device/ in a command prompt:

```
> java -jar VANSampleDevice.jar
```

After executing this command, the window shown in Figure 17 shall appear and the sample device is running properly.

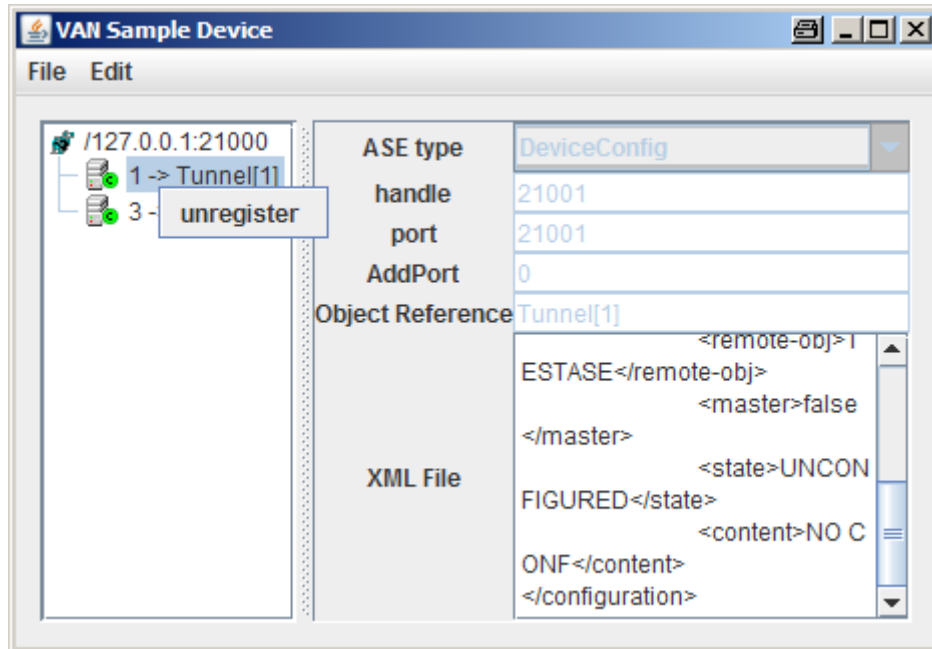


Figure 17: VAN sample device's user interface

The invoked executable runs several processes. Among them there is a device-config ASE that is similar to the one which is run on the DevM machine. It is thus advisable to unregister this ASE from the application registry in order to prevent possible misinterpretations of the results. This can be accomplished by right-clicking the ASE 'Tunnel[1]', which is listed in the tree view (as shown in Figure 17). The option 'unregister' shows up and should then be selected.

4.5 Setup of the Device Machine

The Device Machine should have one network adapter configured for the second IP-subnet. It should be configured with the IP-address '192.168.93.3' and with the '255.255.255.0' subnet mask. As for the APM, a pre-configured version of the sample device provided by WP2 shall be run in order to have a device with Routing ASE and device-config ASE running.

Before starting the sample device, the machine needs to be configured. One of the needed behaviours is the resolution of the names. In order to do this, the following entries have to be added to the POSIX hosts file (<Windows Folder>/System32/drivers/etc/hosts on Windows):

192.168.93.2	ap.dom1.eu
192.168.93.3	dev.dom1.eu

Code Listing 5 - Hosts file for the DevM

After this has been done, the file '<COMMON_SCENARIO>/devm.zip' shall be extracted into an arbitrary folder on the local machine (subsequently mentioned as <DEVM_SW>). This contains a modified version of WP2 implementers' workspace.

The local sample device can be found in '<DEVM_SW>/sample_device'. Its routing table has been adequately configured to route the messages to the APM. The way this has been done

is shown in Code Listing 6, which is an excerpt of the relevant information of the routing configuration file. This file can be found in '<DEVM>/sample_device/2507969.xml'.

```
<TARGET_DOMAIN target-subdomain-name="*.dom1.eu">
  <list-of-next-hop>
    <next-hop next-hop-name="ap.dom1.eu" />
  </list-of-next-hop>
</TARGET DOMAIN>
```

Code Listing 6 - Excerpt of the DevM's Routing Configuration File

Finally, the sample device can be run by issuing the following command line from the directory '<DEVM_SW> /sample_device/' in a command prompt:

```
> java -jar VANSampleDevice.jar
```

After executing this command, the window shown in Figure 17 shall appear, and the sample device is then running properly.

4.6 Deploying the Scenario in a Virtual-Machine Based Environment

The environment that has been described in the previous sections can be easily run on the same physical machine if virtualization software is used. The steps to configure a virtual network with the desired characteristics are described in the following paragraphs. In this description, VMware Virtual Server is used, which is a freely available virtualization software from VMware.

The software can be obtained from <http://www.vmware.com/products/server/>. The installation is straightforward given that no options are changed from their default values.

After it is installed, the computer shall be rebooted. After this has been done, a link to the virtualization software interface can be found in the Windows Start Menu under 'All Programs → VMware → VMware Server → VMware Server Homepage'. If it is selected, the login form of the VMware Server interface shows up in the default web browser, as shown in Figure 18.

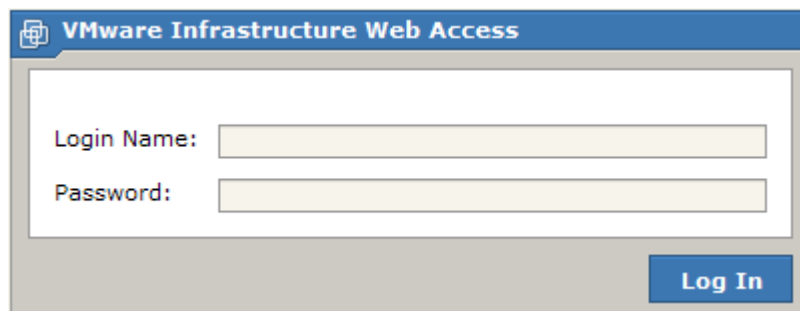


Figure 18: The VMware Server Login Form

Here, the username and password for the operating system's administrator account shall be entered and the VMware Server interface shows up (Figure 19).

The screenshot displays the VMware Infrastructure Web Access interface. The main content area shows the configuration for a virtual machine named 'WXDFC2HZ3JD'. The interface is divided into several sections:

- General:**
 - Hostname: WXDFC2HZ3JD
 - Manufacturer: (empty)
 - Model: (empty)
 - Processors: Intel(R) Core(TM)2 Duo CPU E6550 @ 2.33GHz, 1 CPU x 2 Cores
 - Usage: 232.00 MHz
 - Memory: 1.96 GB
 - Usage: 1146 MB
- Datstores:**

Name	Capacity	Free Space	Location
standard	129.42 GB	67.16 GB	D:\documents and Settings\prolo\My Document
- Networks:**

Name	VMnet	Type
Bridged	vmnet0	bridged
HostOnly	vmnet1	hostonly
NAT	vmnet8	nat
- Commands:**
 - Create Virtual Machine
 - Add Virtual Machine to Inventory
 - Add Datastore
 - Configure Options
 - Edit Host Settings
 - Edit Virtual Machine Startup/Shutdown Settings
 - Refresh Network List
- VMware Tips:**

Move up to production-ready virtualization! It's never been more cost-effective to get a scalable solution. [Learn More.](#)

At the bottom of the interface, there is a task table with columns: Task, Target, Status, Triggered At, Triggered by, and Completed At.

Figure 19: The VMware Server User Interface.

Due to their size, the virtual machines come on a DVD. To get a copy of the DVD please contact the work package leader of WP8 (Friedrich.GOETZ@de.schneider-electric.com). The provided virtual machines can be loaded on the VMware Server by selecting the option 'Virtual Machine > Add Virtual Machine to Inventory'.

After the virtual machines are readily loaded, they can be started by first selecting their entries from the inventory and then pressing the green "play" button. After they have booted, the interaction with them is possible through their "Console" tabs. During the first access to these tabs a prompt to install the "VMware Remote Console Plug-in" is shown in the browser. This can be done by pressing the "Install plug-in" link and following the given instructions.

Finally, the three virtual machines have to be rebooted and the environment will be up and running.

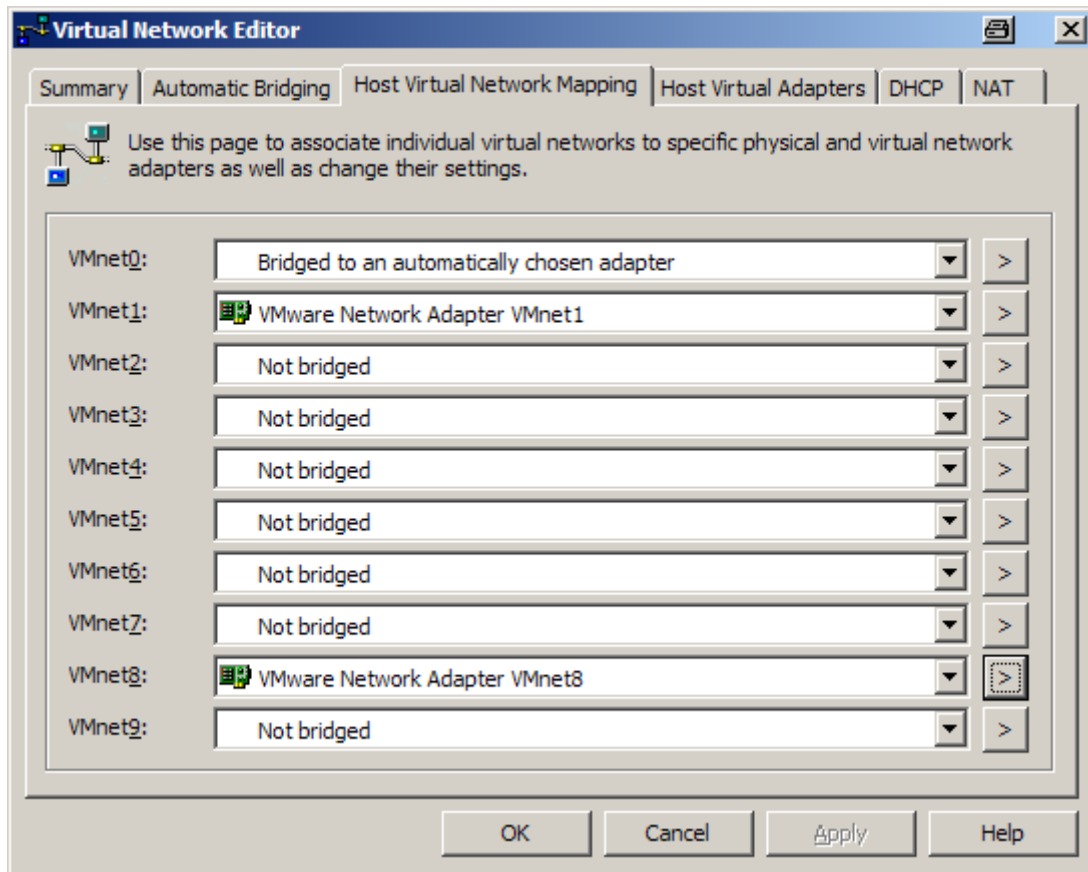


Figure 20: The Virtual Network Editor

It is also possible to run the ECM on the local machine, which is especially useful for development and debugging of the new components of the VAN Engineering Client. In order to replace the virtual machine of the ECM with the local machine, its virtual network adapter needs to be configured properly. For this, the Windows Start Menu entry “Programs → VMware → VMware Server → Manage Virtual Networks” has to be selected and the “Virtual Network Editor” appears as shown in Figure 20. In this window the local virtual network adapter can be configured. The “Summary” tab shows one virtual network, which is marked as “Host-Only”. The subnet for this network adapter has to be configured in the “Host Virtual Network Mapping” tab, just by pressing the little arrow which lays aside of the entry for the network to be configured and by selecting the “subnet” option. The value to be entered as subnet depends on the actual machines. For the ECM it has to be 192.168.92.0 and for the DevM 192.168.93.0. For the APM two different “Host-Only Network adapters” have to be configured with the 192.168.92.0 and the 192.168.93.0 subnets. The second network adapter for the APM can be easily added through the tab “Host Virtual Adapters” by pressing the “Add...” button.

4.7 Testing the Environment

After having the three machines properly configured and the needed applications running, the scenario can be tested by issuing requests from the ECM to the DevM. This not only tests the message sending and reception, but also assures that routing is working properly. Thus, a ‘SetDeviceConfig’ message shall be sent from the ECM’s sample client to the DevM device.

The message to be sent has the DevM device as final destination address. Though, it is first sent to the local sample device, which routes it to the APM’s sample device just as it would do with any message. The APM’s sample device then matches the message’s final destination address with its own routing table entries. According to the configuration the

message is redirected to the appropriate destination: the DevM's device. Then, this device's configuration is changed according to the content of the 'SetDeviceConfig' message and a response is sent back with the ECM as final destination. This final destination's address is first matched against DevM's local routing table, which routes this response to the APM. The APM's sample device does the same, and identifies the ECM as the destination for this message. Finally, the message reaches the sample client of the ECM without passing through the local sample device, because the sample client is listening to port 8080 as all the traversed nodes.

To send the just-described message the sample client that is running on the ECM can be used (Figure 21).

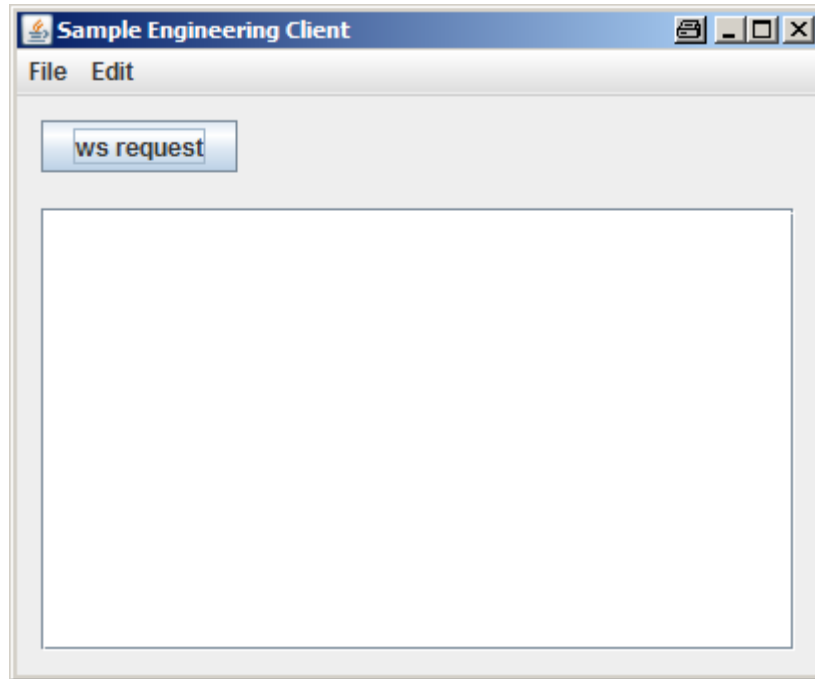


Figure 21: The "Sample Engineering Client" Main Window

The 'ws request' button shall first be pressed. The presentation dialog for the WS-request building dialog appears and the button 'Next' must be pressed. This makes a form appear, where the properties for the recipient and destination addresses have to be filled in. These are presented as 'source' and 'target', and shall be respectively set in the 'WSA-From' and 'WSA-ReplyTo' attributes. The 'WSA-To' field is not configurable, and it is hard-coded with the address of the locally running sample device. The form must be filled in with the values shown in Figure 22 and then the 'Next' button can be pressed.



The screenshot shows a dialog box titled "WS Request Building Wizard Dialog" with a close button in the top right corner. The main title of the dialog is "Addressing".

The dialog contains the following fields:

- URL Scheme:** A dropdown menu with "http" selected.
- source:** A section header.
- host:** A text box containing "ec.dom1.eu".
- port:** A text box containing "8080".
- object reference:** A text box containing "engineering_x".
- target:** A section header.
- host:** A text box containing "dev.dom1.eu".
- port:** A text box containing "8080".
- object reference:** A text box containing "Tunnel[1]".

At the bottom of the dialog, there are three buttons: "Back" (with a left arrow), "Next" (with a right arrow), and "Cancel" (with an 'X' icon).

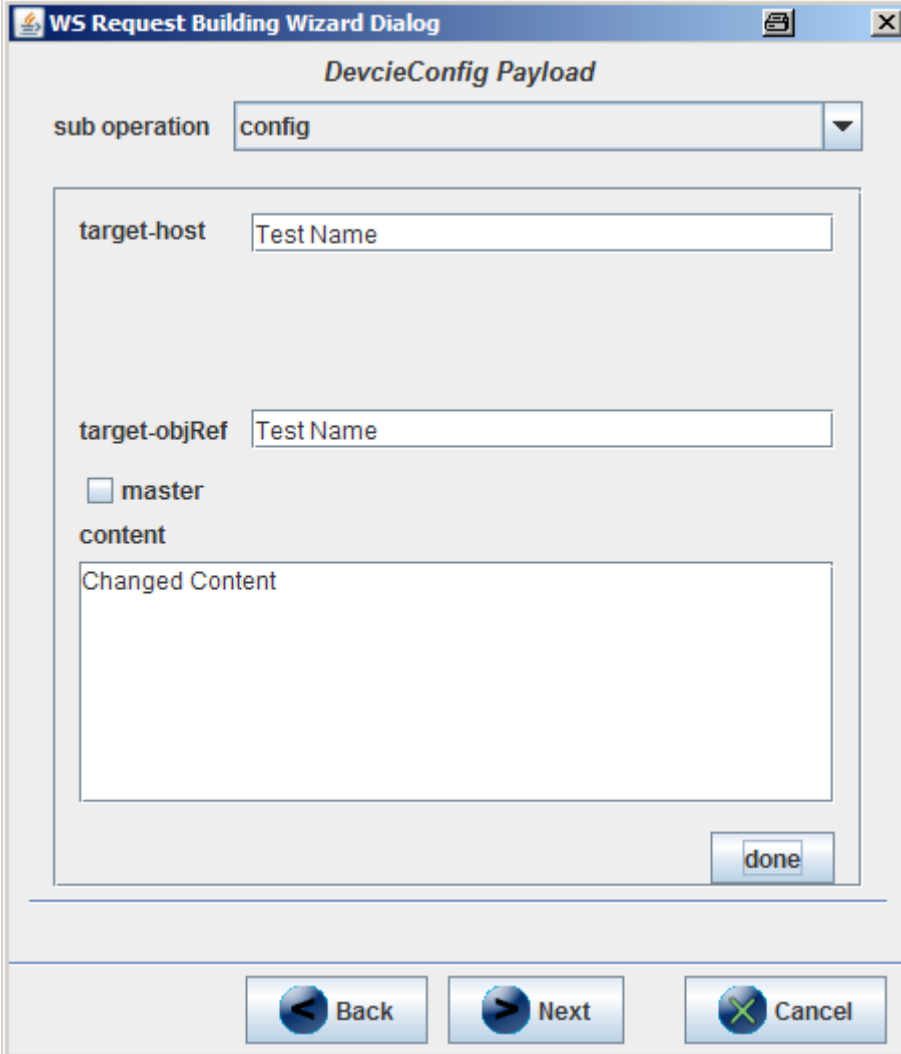
Figure 22: The Sample Engineering Client's Addressing Form

The following displayed form (Figure 23) asks for the call's WS Timeout, WS TTL (number of hops) and the WS Operation to be specified. The predefined WS Timeout and WS TTL values are acceptable for most networks. The only value to be modified is the WS Operation, which has to be set to "SetDeviceConfig" option. Then, the 'Next' button has to be pressed.

The image shows a Windows-style dialog box titled "WS Request Building Wizard Dialog". The main content area is titled "WS Operation". It contains four input fields: "WS Timeout (sec)" with the value "5", "WS TTL (hops)" with the value "3", "WS Operation" with a dropdown menu showing "SetDeviceConfig", and "use VAN Routing" with an unchecked checkbox. At the bottom, there are three buttons: "Back" (with a left arrow), "Next" (with a right arrow), and "Cancel" (with an X icon).

Figure 23: The Sample Engineering Client’s WS Operation Form

Then, the "DeviceConfig Payload" form (Figure 24) is displayed. The "sub operation" shall be set to "config" and the other fields shall be filled with arbitrary data. Afterwards, the 'Next' button can be pressed.



The screenshot shows a Windows-style dialog box titled "WS Request Building Wizard Dialog". Inside the dialog, the title "DeviceConfig Payload" is centered. Below the title, there is a dropdown menu labeled "sub operation" with the value "config" selected. Below this, there are two text input fields: "target-host" and "target-objRef", both containing the text "Test Name". Underneath these fields is a checkbox labeled "master" which is currently unchecked. Below the checkbox is a text area labeled "content" containing the text "Changed Content". At the bottom right of the main form area is a "done" button. At the very bottom of the dialog, there are three buttons: "Back" (with a left arrow), "Next" (with a right arrow), and "Cancel" (with an X).

Figure 24: The Sample Engineering Client Device Payload Configuration Form

The dialog which follows shows a summary of the request that is about to be sent and the 'Finish' button has to be pressed in order to finally send the request.

After some seconds, the "Sample Engineering Client" window displays a message confirming the success of the issued operation (Figure 25).

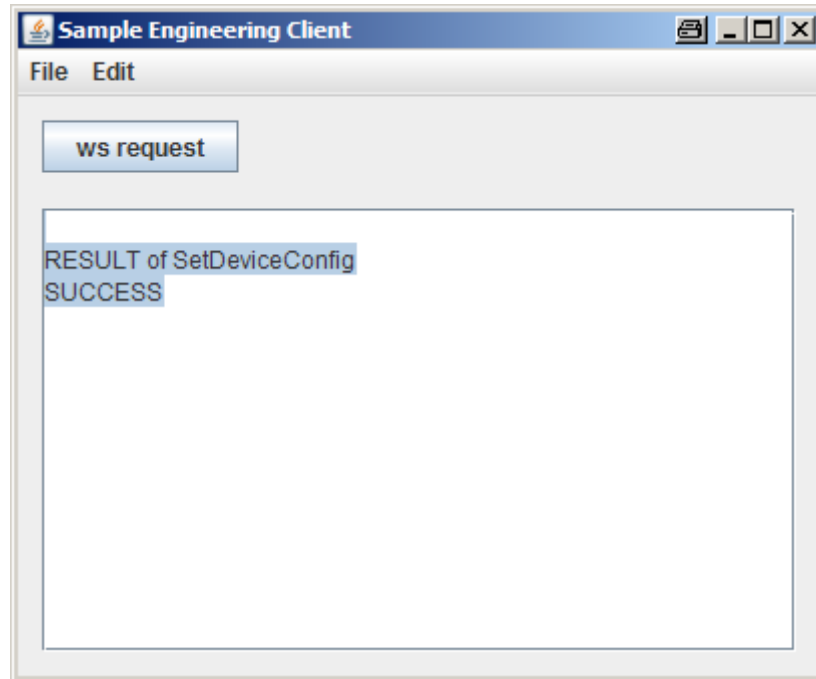


Figure 25: The Sample Engineering Client's Main Window After a Successful Request

The last step is to verify that the message was correctly delivered to the intended device and that the configuration was actually changed on this device. This can be done by checking the "VAN Sample Device" user interface on the DevM (Figure 26). The entry "XML File" for the recently messaged device should contain the content which has been set in the "DeviceConfig Payload" (Figure 24).

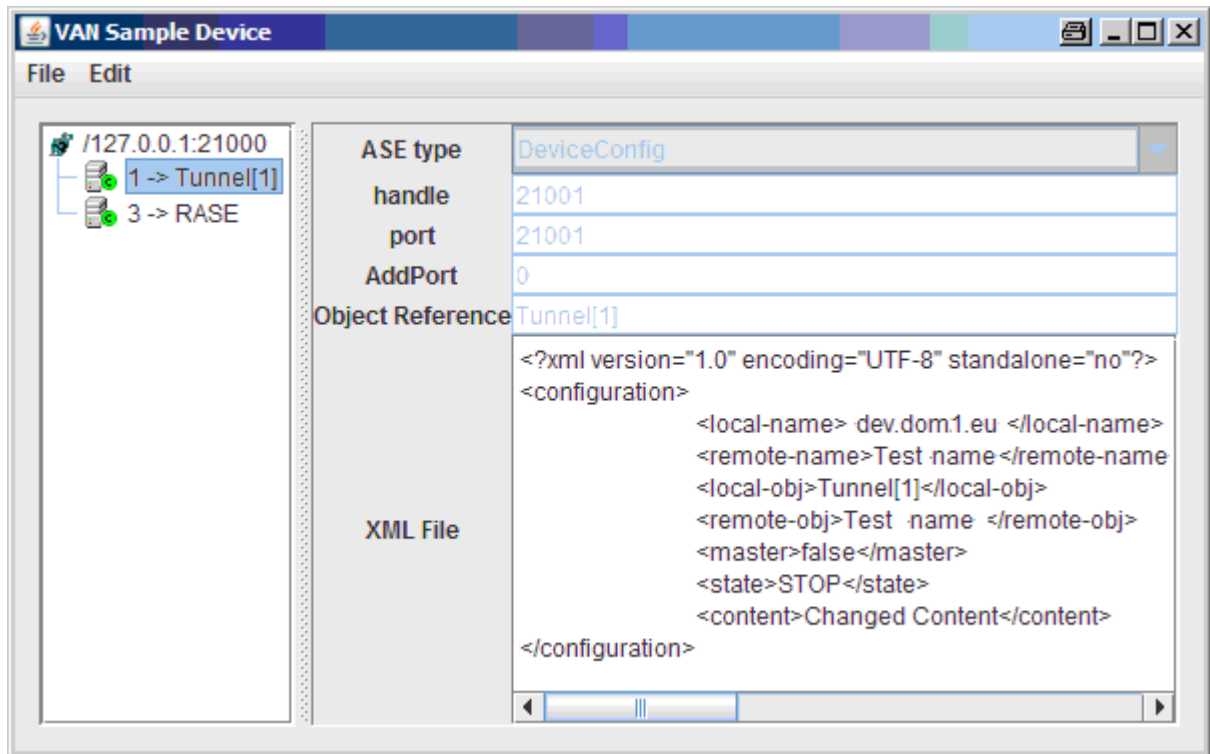


Figure 26: The VAN Sample Device User Interface

5 Enhancements of VAN Engineering Tools

5.1 General Enhancements

5.1.1 Supported ASEs

During the implementation task T8.7 of work package WP8 following requested ASEs are supported by the engineering. The support covers the get and set functionality to configure a VAN Device, the development of a XML Schema for each ASE and the integration of these ASE classes into an example device description (VAN-DD File) for defined test cases. Table 3 lists all supported ASEs.

NOTE: For the Test Scenarios not all classes of the ASEs are implemented and integrated in the IES

Table 3: Supported ASEs

Class ID	Supported Application Service Elements Classes		
	Common ASE Class	Derived Subclass	Derived Sub-Subclass
0x0100	Common_Device_Config		
0x0101		Device_Description	
0x0102		Telecontrol_Config	
0x0103		Link_Config	
0x0300	Common_Domain		
0x0301		Link_Domain	
0x0400	Common_VAN_Peer-To-Peer		
0x0403		QOS_Monitoring	
0x0500	Common_Routing		
0x0501		Target_Domain	
0x0600	Common_VAN_Switching		
0x0602		VAN_Switching_Line	
0x0601		VAN_Switching_Connec tion	
0x0900	Common_Security_Config		
		PF_Device_Security_ Config	
0x0905		VPN_Security_Config	
0x0906			OpenVPN_Security_Config

		ACL_Device_Security_Config	
		Wireless_Device_Security_Config	
0x09A0		VAN_Certificates_PKI	
0x09D0		IDMEF_Client_Configuration	

5.1.2 Reference VAN-DD File for Test Purposes and Support of the IES

For every VAN Device a VAN-DD File must be provided by the vendor. Within the VAN project work package WP8 supports the implementation of VAN-DD Files in terms of the VAN Device prototypes that will be developed for the IES. For this a VAN-DD development guide is available on the VAN groupware [VAN Group]. It provides detailed information how to implement VAN-DD Files and VAN-DD Instance Files.

Further, a reference VAN-DD File has been implemented by work package WP8 for testing purposes, e.g. for VANSAs, VANIT or other prototypes within the project. It contains all ASEs and ASE classes that are implemented as XML schema, see also Table 3. The VAN-DD File is available on the VAN groupware [VAN Group] and located in the following folder:

5.2 Enhancements for VANSAs

5.2.1 Signing of Certificates

The process of signing certificates developed and described in work package WP6 [D06.4-1] and summarised in chapter 2 has to be supported by the VAN Engineering. The aim is to assure a trusted communication path for a secure communication during the bootstrapping of VAN Devices and for the runtime communication between VAN Devices. The resulting task for WP8 is to get and set the necessary certificates via the VAN Certificates PKI class and to provide these certificates in defined directories in a common format to run the certification process.

Therefore, the existing VAN engineering application requires enhancements to assure a secure signing process. For prototypical implementation the Stand Alone Tool (VANSAs) is the chosen tool to support the signing process. The integration of the signing process is described in the workflow below to give a rough overview about the responsibilities of the functions. The signing process itself like mentioned above is done by a batch application. This batch file is provided by work package 6 in close cooperation with work package 8. The advantage of separating these functionalities lies in the independence from specific tools. The VANSAs tool serves only as example for a possible implementation and to show the proof of concept. When the signing of certificates is done, the VANSAs tool prepares the information for integration in the corresponding ASE. After this process the ASE is sent to the VAN Device to finalise the certification process.

The prerequisites for the implementation are:

- available VAN Certificates PKI class as part of the Security_Config_ASE
- valid VAN-DD for a VAN Device that supports security functionalities
- a batch file with command to start the PKI signing process
- Storage on used VAN engineering station with following structure:

[CommonAppDataFolder]/VAN/PKI/temp/UnsignedCertificate.csr

[CommonAppDataFolder]/VAN/PKI/temp/SignedCertificate.crt

Location for all files (Batch, Scripts, Certificates ...):

[CommonAppDataFolder]/VAN/PKI/...

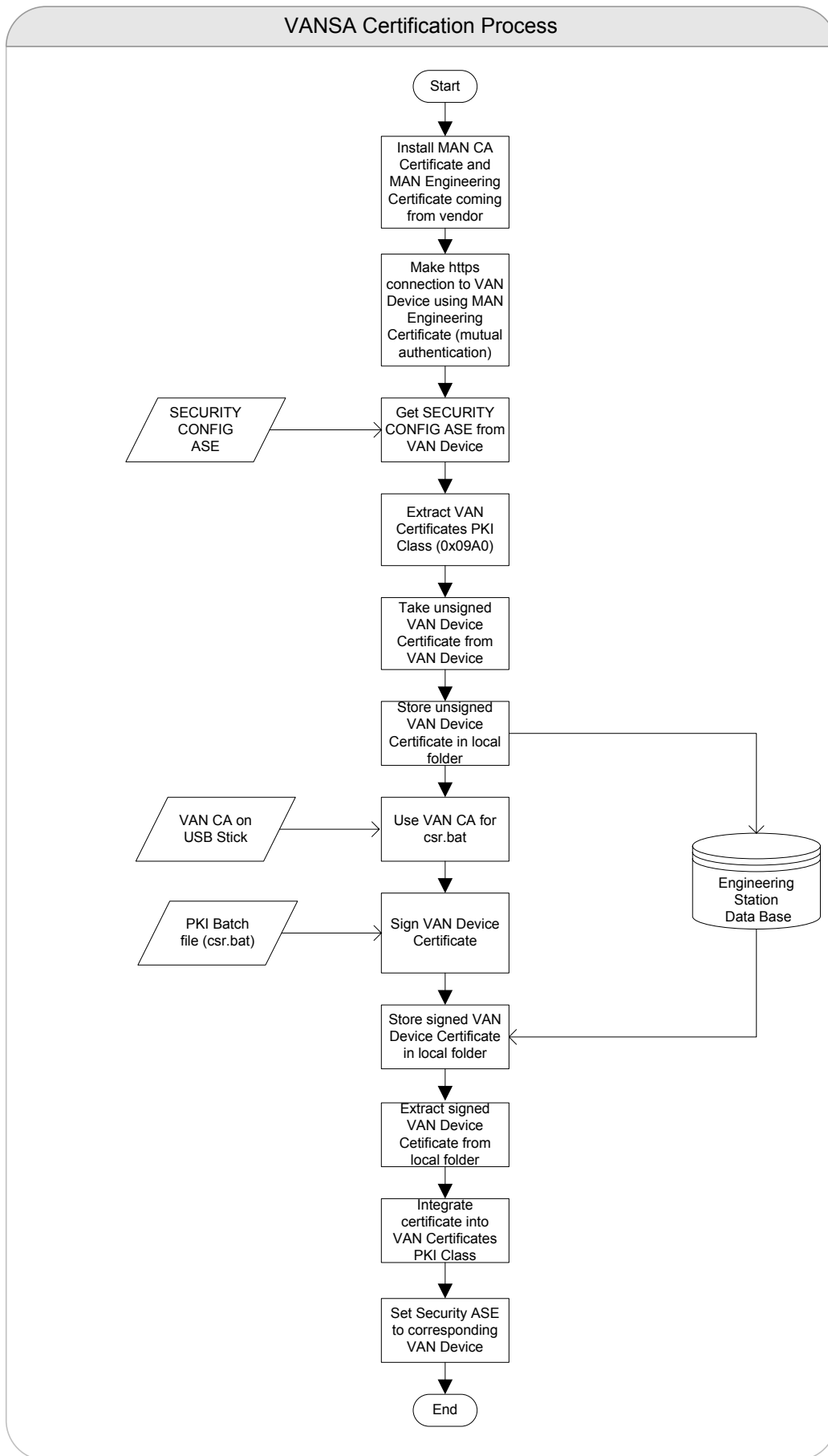


Figure 27: Certification of a VAN Device with VANSAs

To support the engineer who has to certificate new added VAN Devices an easy-to-use mechanism is the best choice. Therefore only one new button for initialising and executing the certification process with its sequence is used like shown in the figure below.

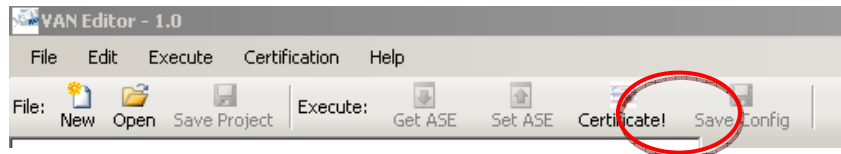


Figure 28: Button for Certification

The task of the engineer is only to assure that all PKI requirements described in [D06.4-1] are fulfilled before he selects the device and pushes the “Certificate” button.

5.2.2 User Authentication

To support the access control as part of the work of WP6 a user authentication is specified. An authentication process is done in several ways, for example by biometric, key or knowledge based user identification. As the most common way and the method with the right degree of security, the password authentication is chosen for authentication in VANSAs. After starting the tool a window is shown where the use has to fill its user name and the corresponding password. If both are valid, the tool can be started.

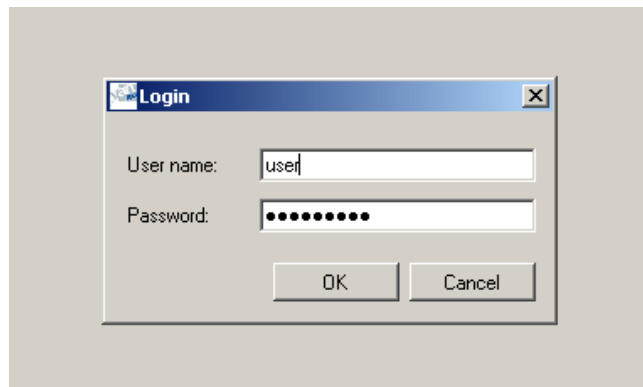


Figure 29: Login for user authentication

For the first implementation only the operator with the right password can use the engineering application. In future versions the authentication is used to personalise the rights what the use can do and what is forbidden.

5.2.3 User Comfort

Within this sub-chapter all enhancement connection with user comfort for the new VANSAs tool release are described. The improvements are based on own experiences, new requirements from other work packages and on self defined enhancements in chapter 3.7 of [D08-5.2]

Report Functionality

For better traceability while using the VANSAs client, the messages within the report window are more detailed. Furthermore the logged reports are stored automatically on the hard disk as txt file in the VANSAs tool directory.

Following messages are new integrated in the report window:

- The user name and password is confirmed
- The current user is “user name”
- Domain „*name of the domain*“ is deleted
- Device “*name of the device*” is deleted
- GET ASE for “*name of the device*” is done
- SET ASE for “*name of the device*” is done
- Certification for “*name of the device*” is done
- “*name of ASE*” is modified

Providing Information

To provide information about a selected domain an additional editable window is available within VANSAs. By selecting a VAN Domain within the tree view, an editable register within the workspace window pops up at the right hand side. Within this window, additional information can be stored.

The figure below shows an example how it is integrated into the tool

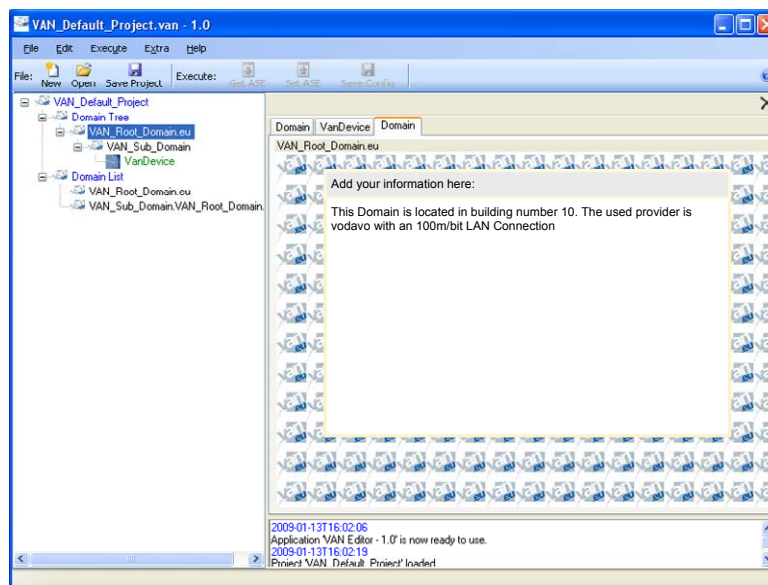


Figure 30: VANSAs Editable Window

User Help

The implementation of help functionality is an essential part for customer support. For the next release of the VANSAs tool a converted description of the VAN Stand-Alone guideline is integrated. The help functionality is started by using the “help” item within the menu bar or pressing “F1”. The new opened window provides the complete VANSAs guideline sorted by categories. With the integrated print functionality menu item “print” the guideline can also be printed.

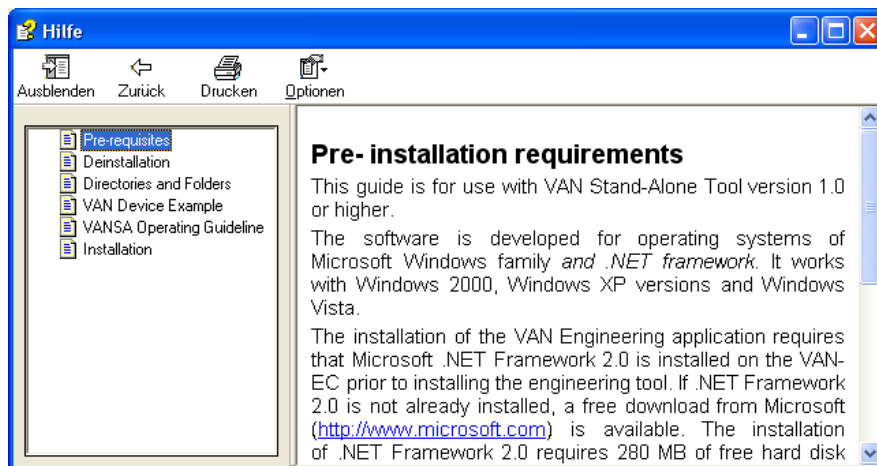


Figure 31: Example of Help Functionality

Save Functionality

The current version of the VANSAs tool has two kinds of saving mechanism implemented as described in [D08.5-2]. One function stores the project file, the other function stores the configuration information in the corresponding VAN DD instance file. This two way approach is improved. Instead of using two buttons for storing the project and the configuration only one "Save" button and one step is necessary to store the complete project including all information within the ASE.

Warranty of Data Integrity

To guarantee unambiguous domain names and device names, a function within VANSAs verifies the given name for a domain or device to be a unique entity in a VAN project.

5.2.4 Support of Secure Transport Layer

The main requirement from WP6 of PKI integration for secure communication is to use https. See the chapters 4.3 and 4.7 and figure 24 and figure 29 for details of accessing a VAN device with https.

5.2.5 Improvements of Rendering Engine

To improve user comfort some minor improvements of the existing rendering engine need to be done. To prevent loss of information a new property has been defined. The new property called AutoStore is has been added to the rendering engine. When this property is set true, the attributes that were changed store their values to the parser before a new node is selected. When the property value is false and a different node is selected, changes will be lost. To allow this new functionality the basic design model had to be extended. The new class diagram of the rendering engine is shown in the Figure 32 and the new class diagram of the tree view is shown in the Figure 33.

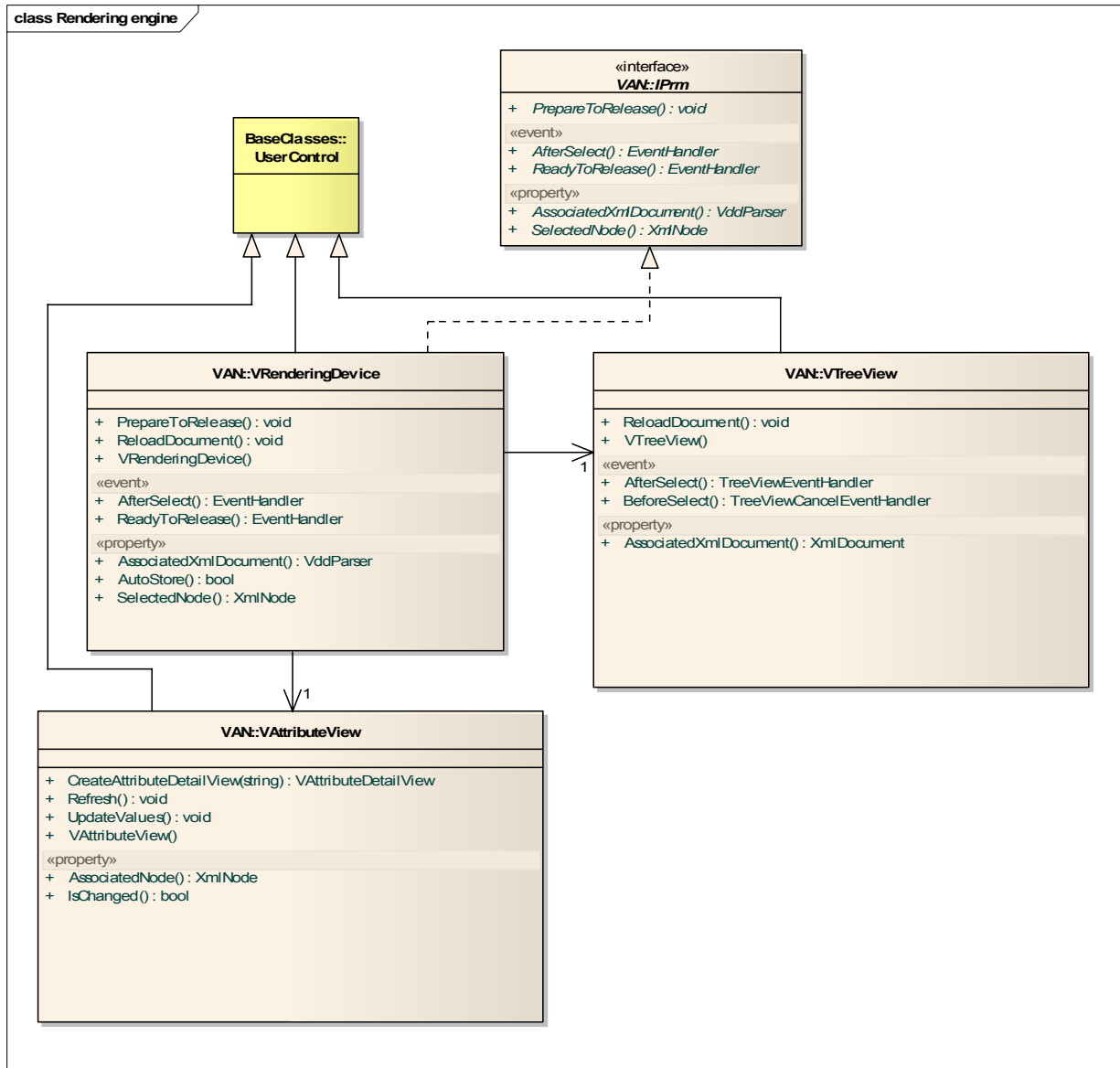


Figure 32: The new extended class diagram of the Rendering Engine

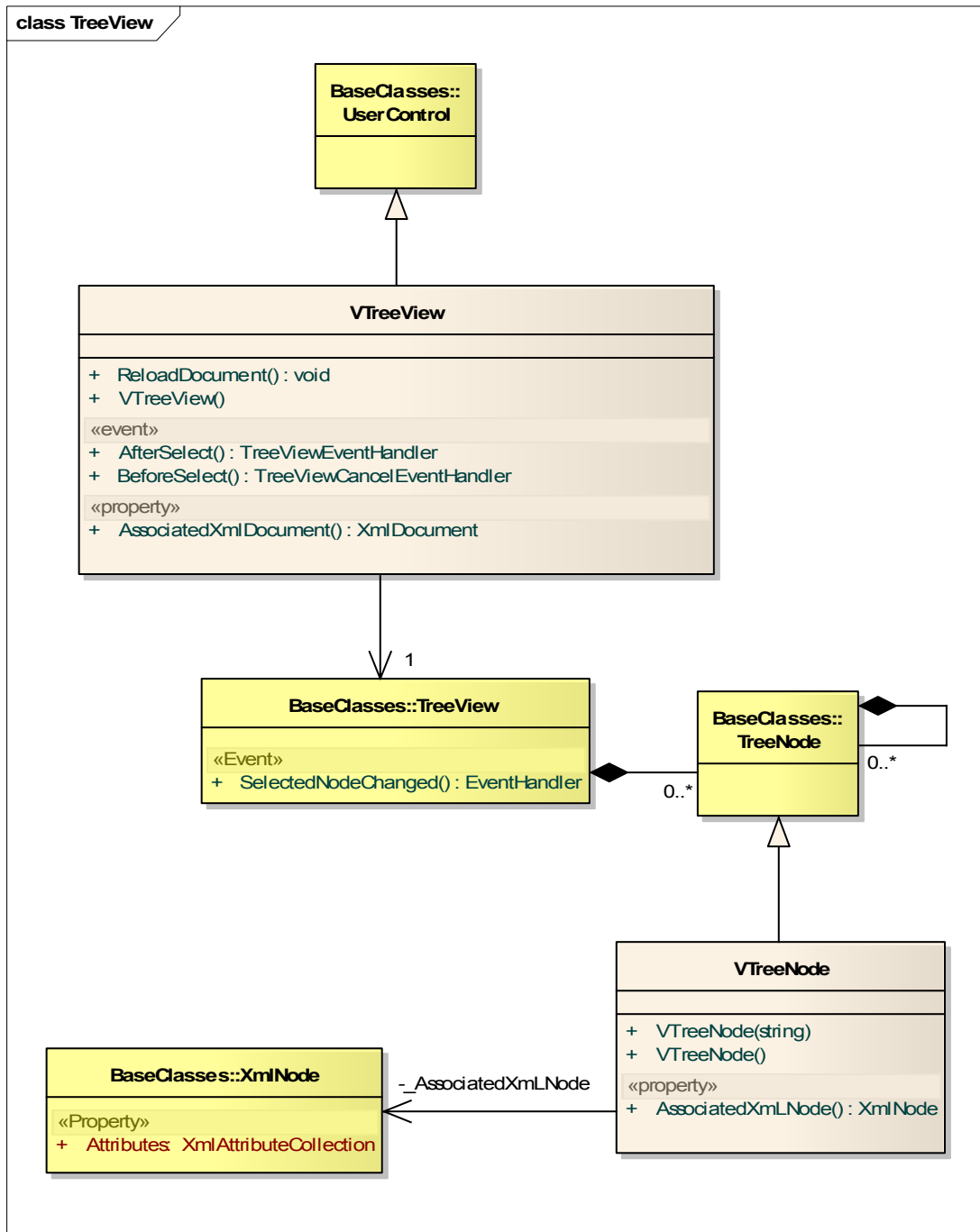


Figure 33: The new extended class diagram of the TreeView

Also some graphical improvements of the rendering engine are done.

All ASE defined and described in the chapter 5.1.1 are implemented to the rendering engine and to the parser.

Improvements of Handling

The following list shows improvements which are pointed out during the work with VANSAs. The improvements are only design and handling features for that a detailed description is not necessary but mentionable.

- The background is adapted to make it more eye friendly

- If an device is chosen from the list the corresponding register within the workspace pops up
- a "remove" functionality is added to be able to delete domains or devices
- After starting the tool a status shows if the Engineering Station is "online" – that means a configuration can be send to the device or "offline" – that means the configuration cannot be sent to the deice and is only stored locally (Link domain as runtime)
- The item "Extra" was deleted, because no further functions are implemented
- After change of an ASE attribute it is not longer necessary to save the changes before switching to another node – changes will not longer deleted

5.3 Enhancements for VANIT

Short motivation and introduce "VANIT"

5.3.1 STEP7

In the local engineering tool STEP7 there is the need to connect the GSD based VAN device to a VAN project file of the Stand-Alone Tool. In the GSD, there is a standard filename for the VAN project predefined. This filename can be changed and stored in the STEP7 project file to allow the use of different DD instance files. The standard path after installation is C:\Documents and Settings\All Users\Application Data\VAN\Projects\VAN_Default_Project.

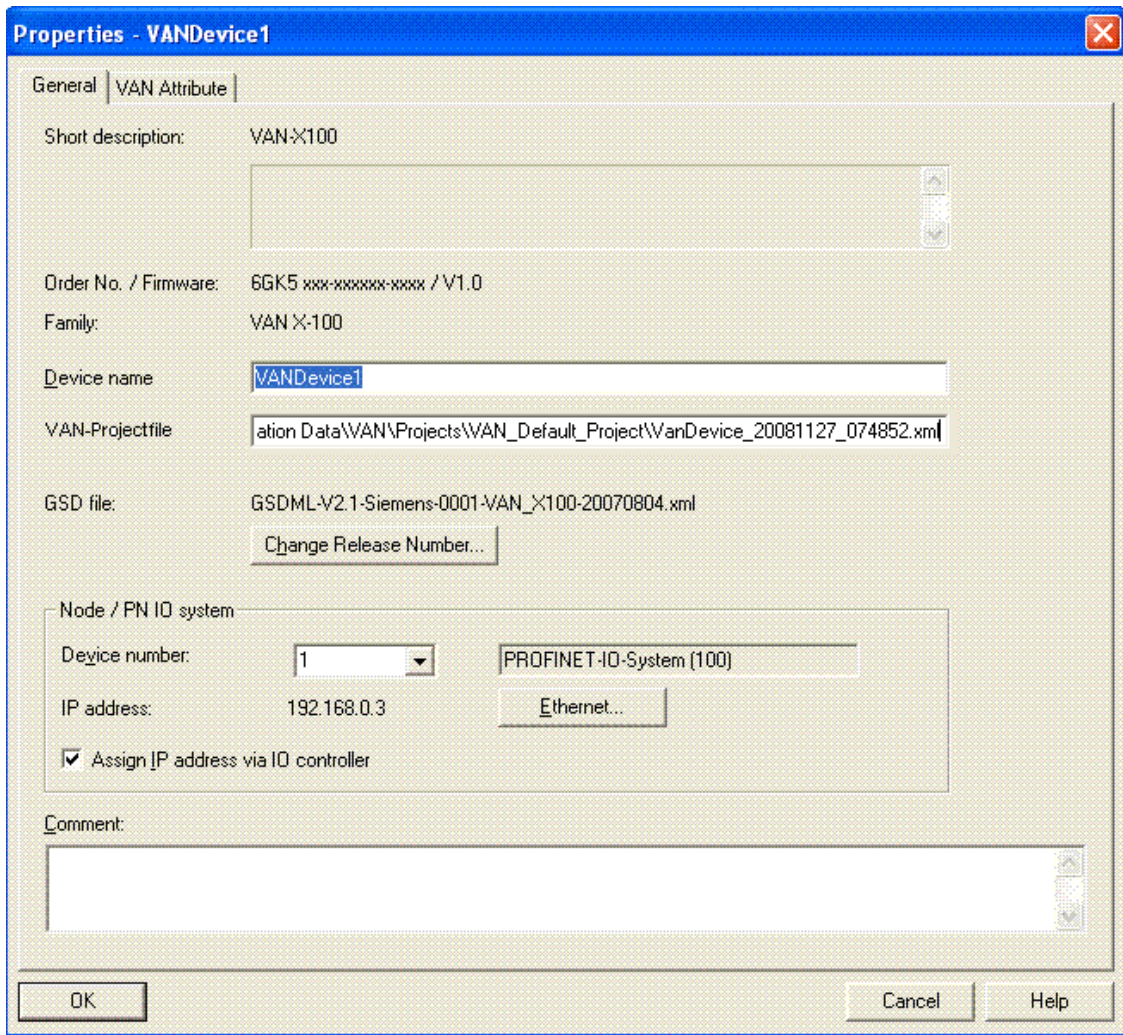


Figure 34: Changeable attribute project file

5.3.2 AutomationXplorer+

In this implementation step of the integrated VAN Engineering Tool, new features regarding security are integrated into the tool. Therefore, the tool itself should also follow the security aspect.

The VANIT prevents unauthorised use of the engineering tool by the password protection of AutomationXplorer+. With this password protection, nobody can start the parameterisation of a VAN Device without knowing the correct password.

The following user administration functions are available in AutomationXplorer+:

Login and logout

To login, select the menu item 'Tools > Login' and enter your user name and password. If another user is logged in when you select 'Tools > Login', this user is automatically logged off. To logout, first close the project and then select 'Tools > Logout'.

Mark the checkbox 'Use Windows login for this user' if you want to login with this user name by default each time you are starting the AutomationXplorer+. In this case, the login is automatically performed while you login to Windows and the password inquiry of the AutomationXplorer+ is skipped on application start.

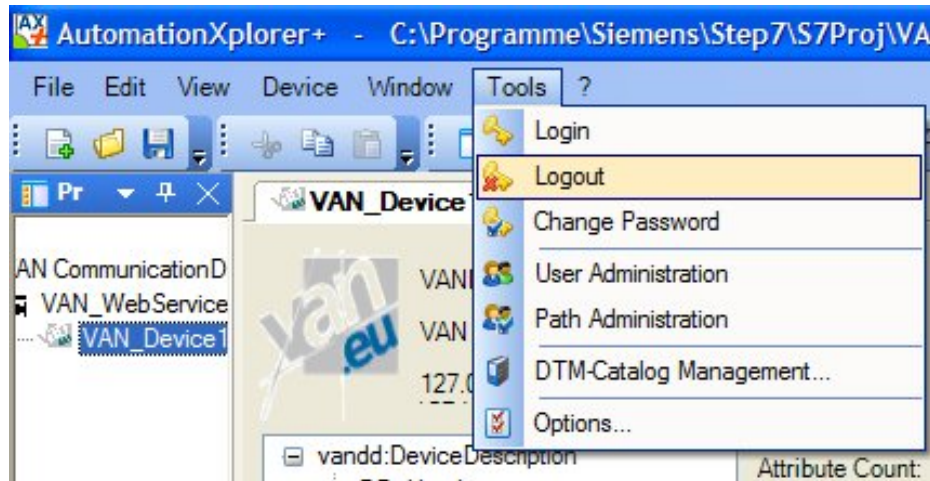


Figure 35: Login function of AutomationXplorer+

Changing the password

For that purpose select the menu item 'Tools > Change Password'. The dialog 'Change Password' appears in which the current user has to enter his current password and his new password twice. After the new password has been entered correctly two times, the button 'OK' becomes active. A graphical display indicates the quality of the password. If the current password has been entered incorrectly, this is indicated after clicking the 'OK' button.

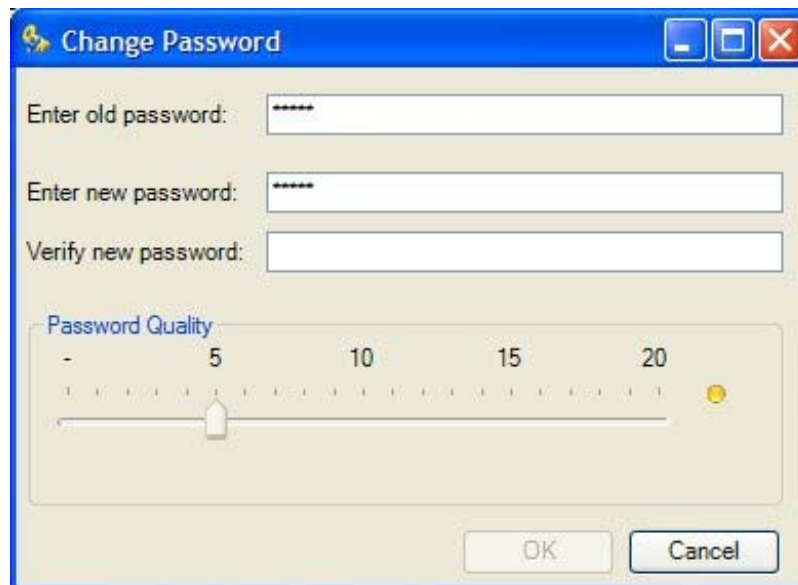


Figure 36: Changing the password of AutomationXplorer+

User Administration

For that purpose select the menu item 'Tools > User Administration'. The following dialogue will appear:

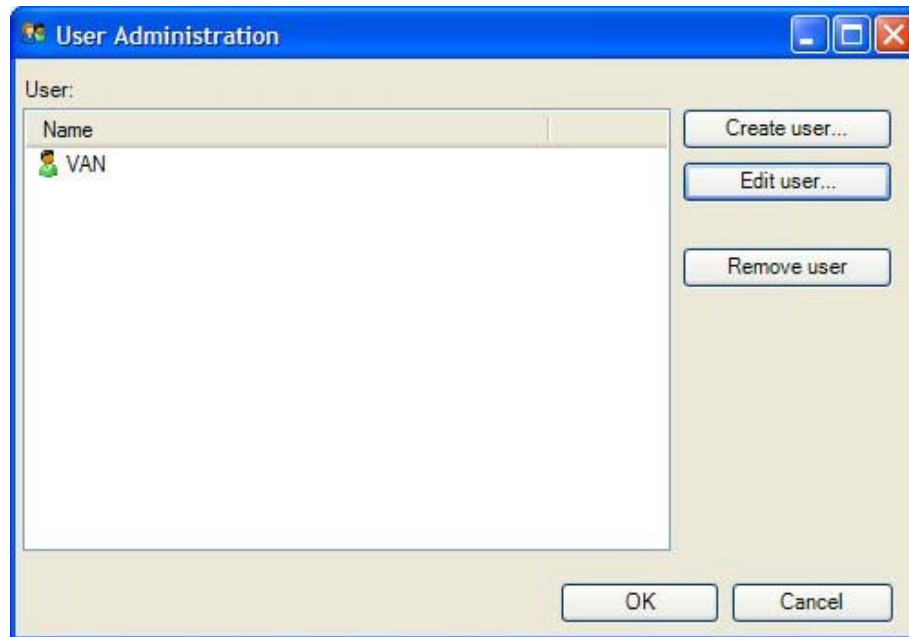


Figure 37: User Administration of AutomationXplorer+

For creating new users and editing existing users, click the button 'Create user...' or mark an existing user and select 'Edit user...' to open the dialogue 'User Properties'.

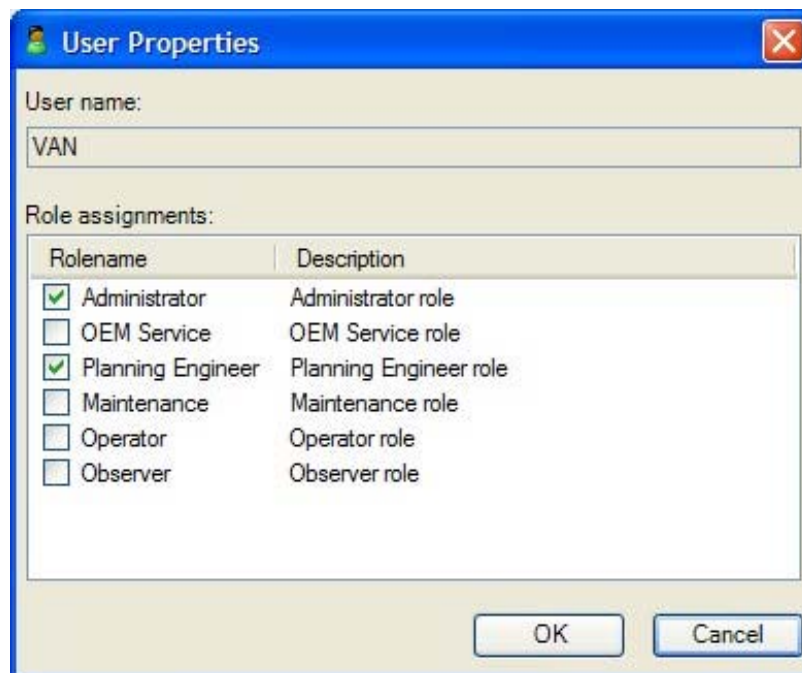


Figure 38: User Properties of AutomationXplorer+

In this dialogue, you can assign the desired access rights to the user. If you have called this dialog using the command 'Create user', you can additionally enter a user name. Otherwise, the user name is write-protected.

For deleting existing users click the button 'Remove user' to delete the user after confirming the appearing security message.

It is not possible to remove the user who is currently logged in.

5.3.3 VAN Device DTM

The new supported ASE classes need also an improvement of the necessary XML schema for the interaction between VAN Device DTM and VAN Communication DTM. These schemata can be generated automatically by means of the VAN Communication Schema Generator 1.0 as already described in [D08.6-1]. From the implementation point of view the detection of the ASE type must be enhanced for the new ASEs and the associated XML schema has to select for the communication.

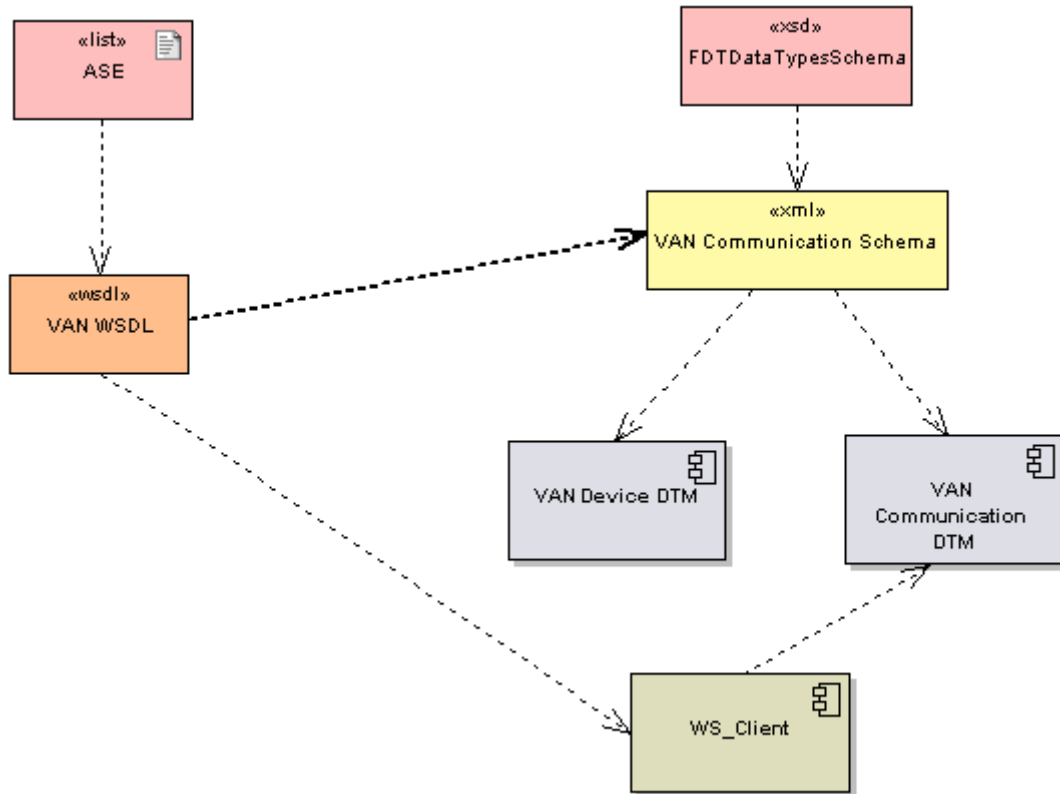


Figure 39: Dependencies of Documents and Components [D08.6-1]

The VAN Device DTM will be improved for using it also without a TCI call. Therefore a FDT project has to be created inside the FDT frame application. In contrast to the recent implementation, the generic VAN Device DTM must be configured before in order to register this DTM for a specific device type. The device type is defined by means of the assigned VAN-DD. This assignment is done at the moment, when the FDT frame application establishes its device catalogue. Inside this procedure, each known DTM is asked for its DTM information (via interface `IDtmInformation`). At this moment, the DTM will browse for all available VAN-DDs and will register itself for the all found device types inside the device catalogue. The path, where all VAN-DDs shall be installed is:

```
$AllUsersAppDataDir$\VAN\DDs
```

The VAN Device DTM publishes via `IDtmInformation.GetInformation()` all device types, which it can handle. The information will be stored inside the device catalogue of the FDT frame application. The information is an association of the user visible device type information and the VAN-DD file. When the user, e.g. the project engineer, would like to instantiate such a device type, the generic VAN Device DTM will be instantiated. The device type specific information are passed via `IDtm.InitNew()` from the FDT frame

application's device catalogue. Then the VAN Device DTM can load the VAN-DD in the same way as the TCI calling approach.

The second difference to the TCI call is the location of the instance data of the DTM. When the VAN Device DTM is instantiated directly from the FTD frame application project, then the VAN Device DTM will save its persistent data also into the repository of the FDT frame application. Therefore no instance data XML file will be created.

The last difference to the TCI calling approach is the assignment of the VAN device address. This is handled completely by the VAN Device DTM via its GUI.

5.3.4 VAN Communication DTM

An overview of the VAN Communication DTM was introduced in deliverable D08.6-1. This overview is recalled in Figure 40.

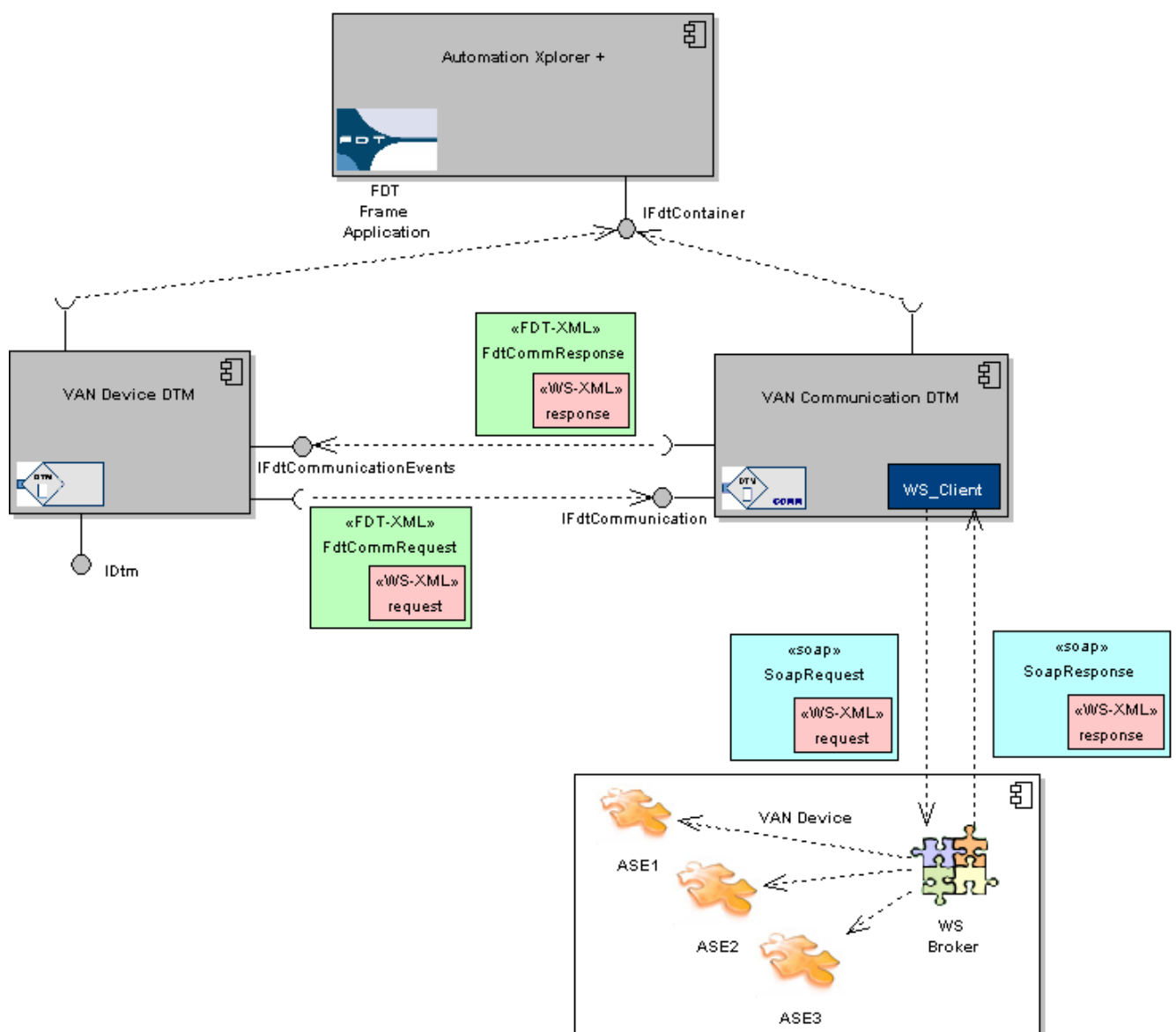


Figure 40: Overview of the VAN Communication DTM

The component *WS_Client* exchanges SOAP messages with the VAN device (blue boxes in Figure 40). In order to support the operations for the newly defined ASEs (see chapter 5.1.1)

and to include the additional header information defined by WS-Addressing (see chapter 3.1) the WSDL for the SOAP messages has been updated.

The code of the *WS_Client* was generated by the tool `wsdl.exe` that can be accessed and executed from the SDK command prompt of the .Net framework [NET_WSDL]. Unfortunately, this framework has not yet implemented the WS-Addressing specification. Therefore, automatic generation of the *WS_Client* is no longer possible.

Moreover, the https protocol (see chapter 2.7) has to be used for the communication to the VAN device. Therefore, the implementation of the *WS_Client* has to be extended.

As described in chapter 3.2 the implementation of the WS client with support of WS-Addressing and https from work package WP2 can be reused by the VAN Engineering Clients. A wrapper of the Java implementation is provided as a C library. Therefore, the integration of the new WS client implementation into the VAN Communication DTM is limited to adaptation from the functions of the *WS_Client* component to the functions provided by the C library.

Additionally, the messages exchanged between the VAN Communication DTM and the VAN Device DTM (green boxes in Figure 40) also depend on the operations of the newly defined ASEs. Therefore, the VAN Communication Schema needs to be updated as well. This, in turn requires that the implementation of the VAN Communication DTM and the VAN Device DTM have to be extended to support the new operations of the updated VAN Communication Schema.

The tool VAN Communication Schema Generator was introduced in deliverable D08.6-1 with the aim to generate the VAN Communication Schema from the WSDL file. As the new WSDL includes some tags introduced by WS-Addressing, the VAN Communication Schema Generator needs to be enhanced in order to accept and correctly interpret the new WS-Addressing tags.

6 Conclusion

The scope of this deliverable was the improvement of the already existing prototypes VANSAs and VANITs regarding the support of work package WP8 for IES.

The enhancements were necessary to fulfil new requests from other work packages, like implementation of new ASE schemas and to be compatible with new technologies used in the VAN devices. The improvements provide proper support for the use cases to be demonstrated in the applications of the Industrial Experimental Setups.

Through the working with final formulated requirements, the planning of the development was stabilized under the restriction of fixed effort to the end of the project.

The basic architectures and interfaces have already been identified in the deliverables [D08.5-1] and [D08.6-1]. These architectures have been extended to integrate new technologies for *security* and *named based routing* in both VANSAs and VANITs.

These new technologies have been prepared in work package WP6, which provided the usage of PKI as a standard technology, and in work package WP2, which provided the usage of *named based routing* also as a standard technology. Also some functional improvements have been made, for example the rendering engine or the GUI of the Stand-Alone Tool.

Also a common development setup for the implementation and testing to be done during the second phase of task T8.7 is used for the improvement of the work at VANSAs and VANITs.

The discussion for improvement was tracked on the requirements document [REQUIREMENTS], where the history of discussion was collected in a comment field.

Therefore the deliverable D08.7-1 is the specification based on the collected requirements of the other work packages for the implementation task D08.7-2.

In the second period of task T8.7 the standards must be integrated and the new technologies must be tested. To handle the implementation comprehensive work for the VAN engineering, the prepared requirement management part *requirement tracking* will be used. The tracking of the requirements collected of the other work packages will mirror the fulfilment of the IES-support.

Glossary

Terminology

Web Server	A communication component provided for the operating system e.g. from Microsoft or Apache foundation
Device	Any device used in an automation system, including network device.
Engineering Client	In the VAN terms, the VAN Engineering Client is a web service client that makes use of SOAP web services. The Engineering Client calls the specific VAN services to perform dedicated parameterization actions.
Engineering Station	In the VAN terms the Engineering Station is a PC which is running an Engineering Client.
Interface	An interface defines the connection of an entity to its environment especially to other entities. It is the basis to create relations between the entities within the described system.
OpenVPN	Software to build tunnel with encryption or without encryption
Project File	A file that is used by the Stand Alone tool to store the engineered information.
Requirement Engineering	Method to collect and achieve the requirements of 14 partners of VAN
Routing	Routing is the process of selecting paths in a network along which to send network traffic.
VAN Device	A VAN device consists of one or more application processes which define the device functionality and the connectivity to a VAN network. See chapter 2.2 of [D02.2-1]. A VAN-DD is a formal description of configuration as well as parameterization capabilities in the VAN context. The specification provides the basic elements to describe a device for the VAN engineering.
VAN-DD File	A VAN-DD File is a XML based file and is used to store the VAN based configuration capabilities for a VAN device. It provides information about supported ASE objects, i.e. the network capabilities from the VAN point of view. It is provided by the vendor of the device. A VAN-DD File can be configured and parameterised respectively within a VAN-ECD. This instantiation is defined as VAN-DD Instance File

VAN-DD Instance File	A VAN-DD Instance File is a XML based file and is used to store the configuration information for a concrete device within a VAN domain. it provides information about supported ASE objects, i.e. the network capabilities from the VAN point of view It is an instance of VAN device description file and can be configured and parameterised respectively within an VAN-ECD.
VAN Domain	Container for all device specific information about the domain, e.g. network topology, infrastructure communication connections; list of other VAN AP's or VAN AD's in the domain

Abbreviations

ACL	Access Control Layer
ASE	Application Service Element
CA	Certification Authority
DD	Device Description
DTM	Device Type Manager
FDT	Field Device Tool
GSD	General Station Description
GSDML	General Station Description Mark-up Language
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
PKI	Public Key Infrastructure
RA	Registration Authority
SSL	Secure Socket Layer
TCI	Tool Calling Interface
TLS	Transport Layer Security
USB	Universal Serial Bus
VAN	Virtual Automation Network
VANIT	VAN Integrated Tool
VANSA	VAN Stand-Alone Tool
VAN CA	VAN certification authority
VAN-DD	VAN Device Description
WSDL	Web Service Description Language
WS	Web Services
XML	Extensible Mark-up Language
XSD	XML Schema Definition

References

- [D02.2-1]. Deliverable D02.2-1 of the VAN project *Specification of the Open Platform for Automation Infrastructure*. The VAN Consortium, 2006
- [D02.4-1]. Deliverable D02.4-1 of the VAN project *V2.0 Software Architecture and Interface Specification*. The VAN Consortium, 2006
- [D02.4-2]. Deliverable D02.4-2 of the VAN project *V1.1 Open Platform and System Architecture, ASE Object Implementation*. The VAN Consortium, 2006
- [D06.3-1]. Deliverable D06.3-1 of the VAN project *Specification of a Security Layer*. The VAN Consortium, 2008
- [D06.4-1] Deliverable D06.4-1 of the VAN project. *Software Architecture and Interface Description*. The VAN Consortium, 2008
- [D08.5-1] Deliverable D08.5-1 of the VAN project. *Specification of Engineering Tool Prototypes for Stand-Alone Concept*. The VAN Consortium, 2008
- [D08.5-2] Deliverable D08.5-2 of the VAN project. *Implementation of Engineering Tool Prototypes for Stand-Alone Concept*. The VAN Consortium, 2008
- [D08.6-1] Deliverable D08.6-1 of the VAN project. *Specification of Engineering Tool Prototypes for Integrated Concept*. The VAN Consortium, 2008
- [JAVA] <http://www.java.com>
- [NET_WSDL] [http://msdn.microsoft.com/en-us/library/7h3ystb6\(VS.71\).aspx](http://msdn.microsoft.com/en-us/library/7h3ystb6(VS.71).aspx)
- [REQUIREMENTS] Requirements_(Template V1.2).xls at WP8 task T8.7 on the groupware server
- [VAN Group] <https://www.van-eu.eu/vangroupware/login.php>