



VAN

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

Virtual Automation Networks

Work Package 8

Engineering of VAN platform
for Embedded Automation Systems

Task 8.4

Engineering Tool Integration Concept
and Tool Interfaces

Deliverable D08.4-2

Specification of object model and tool interfaces
for VAN engineering tool platform

Document type	: Deliverable
Document version	: Final
Document Preparation Date	: 27.12.2007
Classification	: Public
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 and top-level activities	All	04.07.07
0.2	Definition of the main content of the chapters	All	12.07.07
0.3	Integration of all partners contributions for first review phase	Schneider	24.09.07
0.4	Review phase for last draft	All	12.10.07
0.5	Final draft	Schneider	15.10.07
0.6	Final modifications after review from task leader Added Executive Summary, Introduction and Conclusions	Schneider	20.12.07
1.0	Final modifications after review from work package leader	Schneider	27.12.07

Final approval	Name	Partner
Review Task Level	Dr. Harry Hengster	Schneider
Review WP Level	Friedrich Götz	Schneider
Review Board Level	Dr. Axel Klostermeyer	Siemens

Executive Summary

This document is the deliverable D08.4-2 of the VAN project and comprises the results from the activity of the participants within task T8.4 “Engineering Tool Integration Concept and Tool Interfaces” of work package WP8 “Engineering of VAN platform for Automation Systems”. It is a result of cooperation of the parties: University of Magdeburg (CVS), Ifak Magdeburg (Ifak), Schneider Electric (Schneider), Siemens AG (Siemens) and Phoenix Contact (Phoenix).

The *Introduction* starts with the description of the main objectives of task T8.4 and motivates the distribution of the content of the different topics in the two documents D08.4-1 and D08.4-2 of this task. It also recalls the relevant topics of previous deliverables of work package WP8, for which details are worked out in the subsequent chapters of the deliverable at hand.

Chapter *VAN Engineering Tool Use Case* specifies concrete use cases with the steps to be performed in the integrated concept and in the stand-alone concept. These use cases form the basis to become aware of the expected user experience for the engineering tool as well as to derive and validate an accurate specification of the required interfaces.

Chapter *VAN Information Model* provides an update of the VAN information model with respect to the newly defined application service elements (ASEs) and some additional relations, which have been discovered after completion of deliverable D08.3-1. Moreover, the design for a decomposition of the VAN Instance Model into different files is developed. The VAN Instance Model is described for an example of a VAN system and the corresponding UML model is presented.

Chapter *VAN Device Description* specifies a generic XML schema for VAN device descriptions (VAN-DDs). The XML schemas are also designed based on a decomposed approach in order to allow for easy extension for new ASEs, which will be defined in future. It provides a guideline describing how ASEs and ASE classes have to be defined and how they have to be integrated into the existing schemas. Additionally, it describes how a VAN-DD can be instantiated for a concrete VAN device and how the actual parameter values for a VAN device can be stored in a VAN-DD instance file.

The next two chapters provide technical specifications for the technologies *Tool Calling Interface* (TCI) and *Field Device Tool* (FDT), which are used for the integrated concept.

In chapter *VAN Tool Calling Interface*, a concrete definition of a Program Interface Description (PID) is given for the integration of a Device Tool in the menu structure of an Engineering System according to the TCI specification. Moreover, the architecture of TCI and the interrelationship with the Temporary Parameter File (TPF file) are explained.

Chapter *VAN Field Device Tool* describes the interfaces, the user roles, and the operation phases of the FDT specification. The difference between the centralistic, user role oriented approach of FDT and the distributed, certificate oriented VAN Access Control Layer (ACL) defined in work package WP6 is elaborated and a mapping between the two approaches is proposed. To describe the interfaces of the Device Type Managers (DTMs), the Web Services Description Language (WSDL) file to be used for communication between the VAN device and the VAN Communication DTM is presented and the VAN Communication schema for the data exchange between the VAN Communication DTM and the VAN Device DTM is specified.

In the *Conclusions* the purpose and main results of the deliverable are summarized. Moreover, the correlation to other work packages as well as to the other tasks of work package WP8 is pointed out.

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

Contents

1 Introduction	7
2 VAN Engineering Tool Use Case	9
2.1 Integrated Concept	9
2.1.1 Operations within Step7	9
2.1.1.1 Import VAN Instance Model part of a VAN Device	10
2.1.1.2 Add a new VAN Device to Profinet	10
2.1.1.3 Send and Request Transfermodules for VAN	13
2.1.1.4 Send or Request Transferunits for VAN	15
2.1.2 TCI Call	16
2.1.2.1 Context Menu through TCI	17
2.1.2.2 Reading VAN Device Description	17
2.2 Stand-Alone Concept	17
2.2.1 Create and Select a VAN Domain	18
2.2.2 Create/Select a VAN Sub Domain	19
2.2.3 Build up Interconnections between VAN Sub Domains	19
2.2.4 Add New Device	19
2.2.5 Configuring new VAN Devices	20
2.2.5.1 Read ASEs from Device	21
2.2.5.2 Demonstration Implementation	22
2.2.5.3 Write ASEs to the Device	22
2.3 Storing of a VAN Project	23
3 VAN Information Model	24
3.1 Update of the Information Model	24
3.2 VAN Instance Model	25
3.2.1 Characteristics of the Use Case	25
3.2.2 Design of the VAN Instance Model	30
4 VAN Device Description	32
4.1 VAN Device Description Schema	32
4.1.1 Used XML Schema Elements	33
4.1.2 Rules and Naming Conventions for Schema Elements	33
4.1.3 Update and Extension	34
4.1.4 Generic Structure of the VAN-DD Schema	34
4.1.5 Implementation of new ASEs or ASE Classes	36
4.1.6 Naming Conventions of a VAN Device Description File	39
4.2 VAN-DD for VAN Devices	39
4.3 VAN-DD Instance File for a VAN Device	40
4.4 GSDML for VAN Device	41
4.4.1 Introduction to GSD	41
4.4.2 Structure of a GSD	41
4.4.3 Useful parts for VAN Devices	41
5 VAN Tool Calling Interface	41
5.1 Program Interface Description for VAN	41
5.1.1 Example of a PID File	41
5.2 Architecture of TCI	41
5.3 VAN Temporarily Parameter File	41
5.4 General Structure Definition	41

6 VAN Field Device Tool 41

6.1 VAN Device DTM 41

6.1.1 Supported Interfaces 41

6.1.2 User Roles 41

6.1.3 Relationship between User Roles and ACL 41

6.1.4 Representation of ASE 41

6.1.5 Operation Phases 41

6.2 VAN Communication DTM 41

6.2.1 VAN Web Services and WSDL 41

6.2.2 SOAP in VAN 41

6.2.3 VAN Communication Schema 41

6.2.4 VAN Communication DTM Sequence Diagram 41

7 Conclusion 41

Glossary 41

Terminology 41

Abbreviations 41

References 41

List of Figures

Figure 1: Example of the using of VAN Devices at Profinet	10
Figure 2: Adding a Profinet Device by Drag and Drop to IO-System	11
Figure 3: Editing Attributes for VAN Devices.....	12
Figure 4: Access to the Profinet Part of a VAN Device	13
Figure 5: Send and Request Transfermodule	14
Figure 6: Send or Request Module with VAN Address	15
Figure 7: Send or Request Transferunits	16
Figure 8: Approach for a Stand-Alone Concept	18
Figure 9: Engineering of Communication Connections - Concept 1	20
Figure 10: Engineering of Communication Connections – Concept 2.....	21
Figure 11: Control Elements of the GUI	22
Figure 12: Screenshot of ASE View	23
Figure 13: VAN Network View.....	24
Figure 14: Use Case Scenario	25
Figure 15: Instance Files	30
Figure 16: UML Representation of VAN Instance Model	31
Figure 17: Specification of new ASE Class Schemes	34
Figure 18: VAN-DD Main Elements.....	34
Figure 19: VAN-DD Header Element	35
Figure 20: Supported ASEs Element	36
Figure 21: Main Elements of Device Config ASE	36
Figure 22: Inclusion of an Imported Element.....	37
Figure 23: Example- VAN Device Config ASE	38
Figure 24: Instantiation of a VAN Device Description File.....	39
Figure 25: Example Schema of a VAN Device Description File.....	40
Figure 26: VAN-DD Instance File	41
Figure 27: VAN-DD Schema References GSD	41
Figure 28: The Top Level Structure.....	41
Figure 29: Context Menu through PID File.....	41
Figure 30: TPF File Structure	41
Figure 31: Schema of a Public Key Infrastructure.....	41
Figure 32: User Interface of a VAN Device DTM	41
Figure 33: Role of the VAN Communication DTM.....	41
Figure 34: VAN Integrated Concept Architecture	41
Figure 35 : Communication between VAN-EC and VAN Device.....	41
Figure 36: WSDL file for the VAN Prototype	41
Figure 37: Architecture of the FDT Communication	41
Figure 38: VAN Communication DTM Sequence Diagram	41

1 Introduction

The results of task T8.4 are provided in two deliverables. The general concept for the engineering of VAN related attributes in a VAN system has been developed in the first phase of the task and the corresponding results have been provided in deliverable D08.4-1. Two concepts for the realization of the VAN related enhancements have been introduced in that deliverable.

On the one hand, the stand-alone concept was proposed. This concept can be used to develop a new tool, which covers the role of a VAN-ECS [D08.2-1] for modelling and planning of a complete VAN system as well as the role of a VAN-ECD [D08.2-1] for configuration and parameterization of a VAN device [D08.3-1]. VAN Engineering Tools following the stand-alone concept are independent from existing engineering tools used for other engineering tasks of automation systems. Therefore, no changes are necessary for existing engineering tools. A component-based design for the stand-alone concept was already partly implemented in an Engineering GUI [D08.4-1], which was successfully used for configuration of ASEs [D02.1-1, D02.2-1] in a first prototype of a VAN system.

On the other hand, the integrated concept was introduced. This concept focuses on the integration of the necessary VAN enhancements into existing engineering tools. In order to facilitate the integration of VAN specific features the standardized interfaces for FDT/DTM [FDT1.2.1] and for TCI [TCI] are used. By use of the FDT technology, it is possible to integrate VAN specific Device Type Managers (DTMs), which can be developed in the scope of the VAN project and may be used by any vendor. The TCI technology allows coupling of different engineering tools. Using these established and public available standards for the integrated concept allows a smooth integration in the existing automation tool market.

In deliverable D08.4-2 use cases are specified for both concepts, in order to clarify the expected behaviour and workflow from the users perspective. These use cases are also used to derive the specification of the interfaces between the different components in the stand-alone and the integrated concept. Since the existing engineering tools Step7 from Siemens and AutomationExplorer+ from Phoenix will be used for the implementation of a prototype of the integrated concept, the use cases for this concept are focused on the user interface and features of these tools. As Step7 is dedicated to Profinet, this also implies that the use cases for the integrated concept will be based on Profinet. Since Profinet is specified as Type 10 of the public standard IEC 61158 [IEC 61158] the presented use cases can be extended to other fieldbus types as well. Based on the presented use cases technical specifications for the application of the FDT interfaces and the TCI interface in the context of the necessary enhancements for VAN Engineering Tools are devised and concrete examples are given.

Besides the specification of the interfaces, the objective of task T8.4 is also to describe the object models to be used. The VAN Information Model and the concept for the VAN device description (VAN-DD) have been introduced in deliverable D08.4-1 and they are detailed and improved in deliverable D08.4-2.

In addition to the updates of the VAN Information Model, which were necessary to incorporate the emerged results from other work packages and to reflect some newly discovered relations, deliverable D08.4-2 also introduces a decomposed approach for the VAN Instance Model. While the VAN Information Model describes the elements and structure of a VAN system, the purpose of the VAN Instance Model is to represent a concrete instance of a VAN system. Therefore, the actual elements and the relations between the elements as well as all values of the different attributes of the complete VAN system have to be described. The original idea as described in deliverable D08.4-1 was to put any needed information of a complete VAN system into one single instance model. However, during the technical design in D08.4-2 it turned out that this approach would result in a unwieldy model and it would as well include some redundant information, which is also covered by the device description files. Therefore, the VAN Instance Model covers only the structure and elements of the overall VAN system but not the detailed information for the device configuration.

For the description of the VAN device configuration the basic concept of the VAN-DD was presented in D08.4-1. The actual schema definition for the VAN-DD is developed in this deliverable. To allow for easy extension of the schema for newly defined and enhanced ASEs and to ease the creation of VAN-DD for any VAN devices, which supports some subset of defined ASEs, a decomposed approach is used also for the definition of the VAN-DD schema. Based on this schema definition a VAN-DD instance file is introduced, which can be used to provide the actual parameter values for the VAN devices of a VAN system. The VAN-DD instance files complement the information of the VAN system. Therefore, they are referenced by the VAN Instance Model.

To validate the design of the updated VAN Information Model and the schemas for the VAN-DD examples are defined in the deliverable and the corresponding VAN Instance Model and VAN-DD instance file, respectively, are presented. Moreover, a guideline is provided, which describes how VAN-DD schemas for new ASEs and ASE classes have to be defined and how they have to be integrated into the existing schemas. Additionally, it describes how a VAN-DD can be instantiated for a concrete VAN device and how the actual parameter values can be stored in a VAN-DD instance file.

2 VAN Engineering Tool Use Case

2.1 Integrated Concept

Before operations within a local engineering tool are defined, it is necessary to explain the environment of the local engineering tool. For example in Step7 the Simatic Manager is used for the local engineering. When a new project is created as a VAN project then the Simatic Manager is used to define which devices and which protocol type, e.g. Profibus or Profinet, are required for a specific controller.

The integrated concept is a proposal for the engineering of a VAN device. This integration of Step7 into the VAN architecture is shown as one possible implementation of the integrated concept. A VAN device is connected with Profinet standard to the local environment and to the VAN network segment (see [D08.4.1]) with application service elements (ASEs). To use these standards, the implementation is easier, because many operations can be done by already implemented mechanisms at those standards. For example, the engineering in Step7 is automated through the data in the Generic Station Description (GSD). With the help of the GSD the user is forced to use the modules provided by the GSD and cannot make mistakes.

2.1.1 Operations within Step7

This chapter shows a possible integration of an existing, local engineering tool into the VAN architecture. For the integrated concept the VAN Instance Model must be imported, then a VAN device must be selected and the parameters for the ASEs must be set. The infrastructure of the local engineering tool for example Step7 is using Profinet. Step7 is importing the VAN Instance Model which provides network and device information of this local sub domain. For detailed information about the VAN Instance Model, see chapter 3.2.

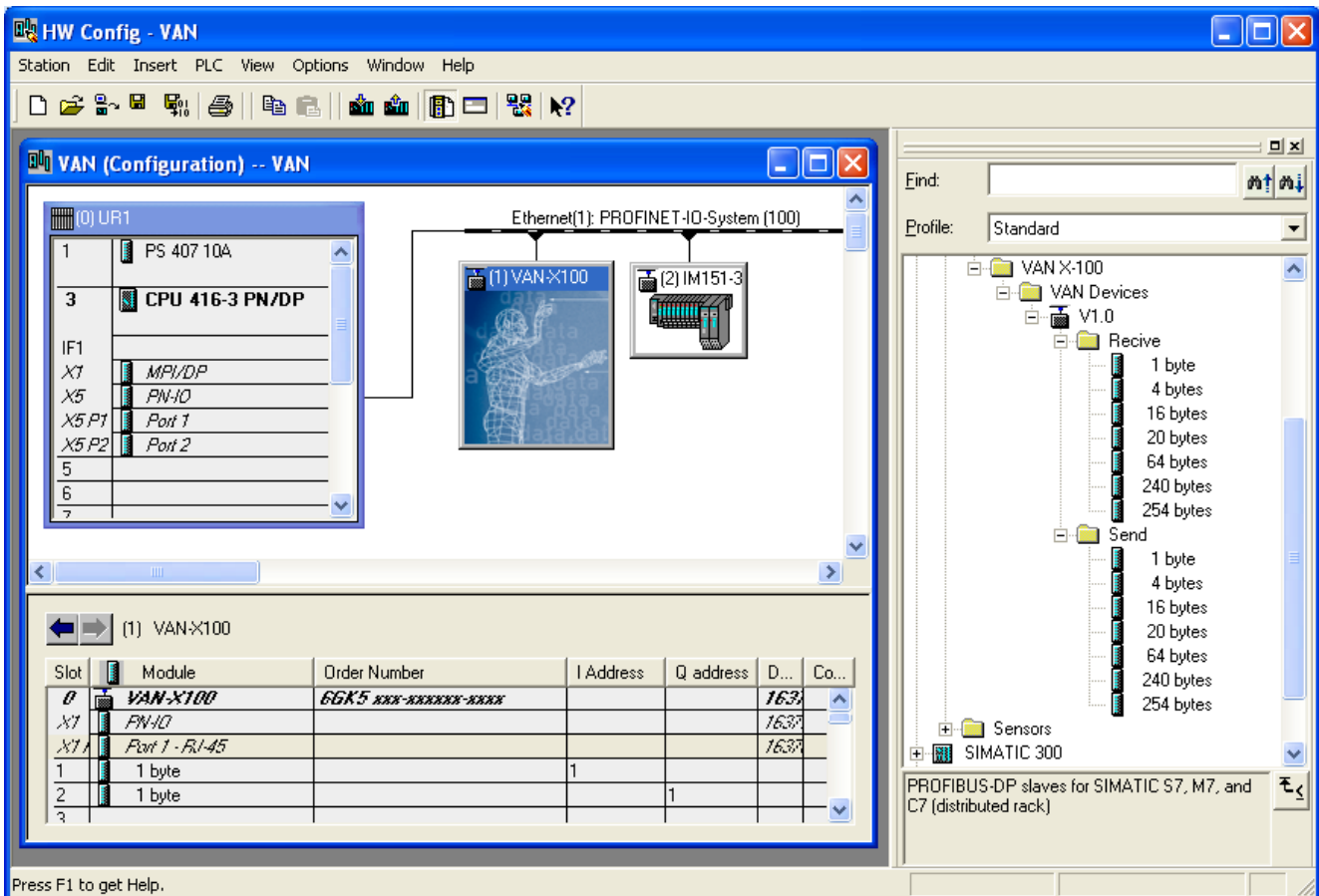


Figure 1: Example of the using of VAN Devices at Profinet

Figure 1 shows a VAN device, which is providing the automation data of a device named IM151 in the picture included through the Profinet for the VAN network segment.

2.1.1.1 Import VAN Instance Model part of a VAN Device

Each VAN device is configured in a VAN system. The configuration parameters are stored in the VAN Instance Model, which is conform to the VAN Information Model, which is defined in the VAN-ECS [D08.2-1]. The VAN Instance Model and the VAN devices description instance file (VAN-DD instance file; see chapter 4.3) provide the complete VAN system configuration. Thus, one VAN device configuration is a XML-part of the XMI file. When the VAN Instance Model is available, e.g. through the VAN-ECS, the local engineering tool reads the XMI file to get the necessary parameters and information of the remote VAN device. It must be ensured that the GSDs for all devices in the VAN Instance Model are available in STEP7.

2.1.1.2 Add a new VAN Device to Profinet

If we have a VAN device in the Step7 hardware catalogue, installed by a GSD, the user can use Drag and Drop to bring a VAN device to the Profinet. In the VAN Engineering Tool architecture an XMI file which represents the VAN Instance Model, defines which devices could be used. For example, the local engineering tool imports this XMI file and offers the available VAN devices. Figure 2 shows a Profinet-IO System where the data of ETS200s modules are provided through a VAN device [IEC 61158].

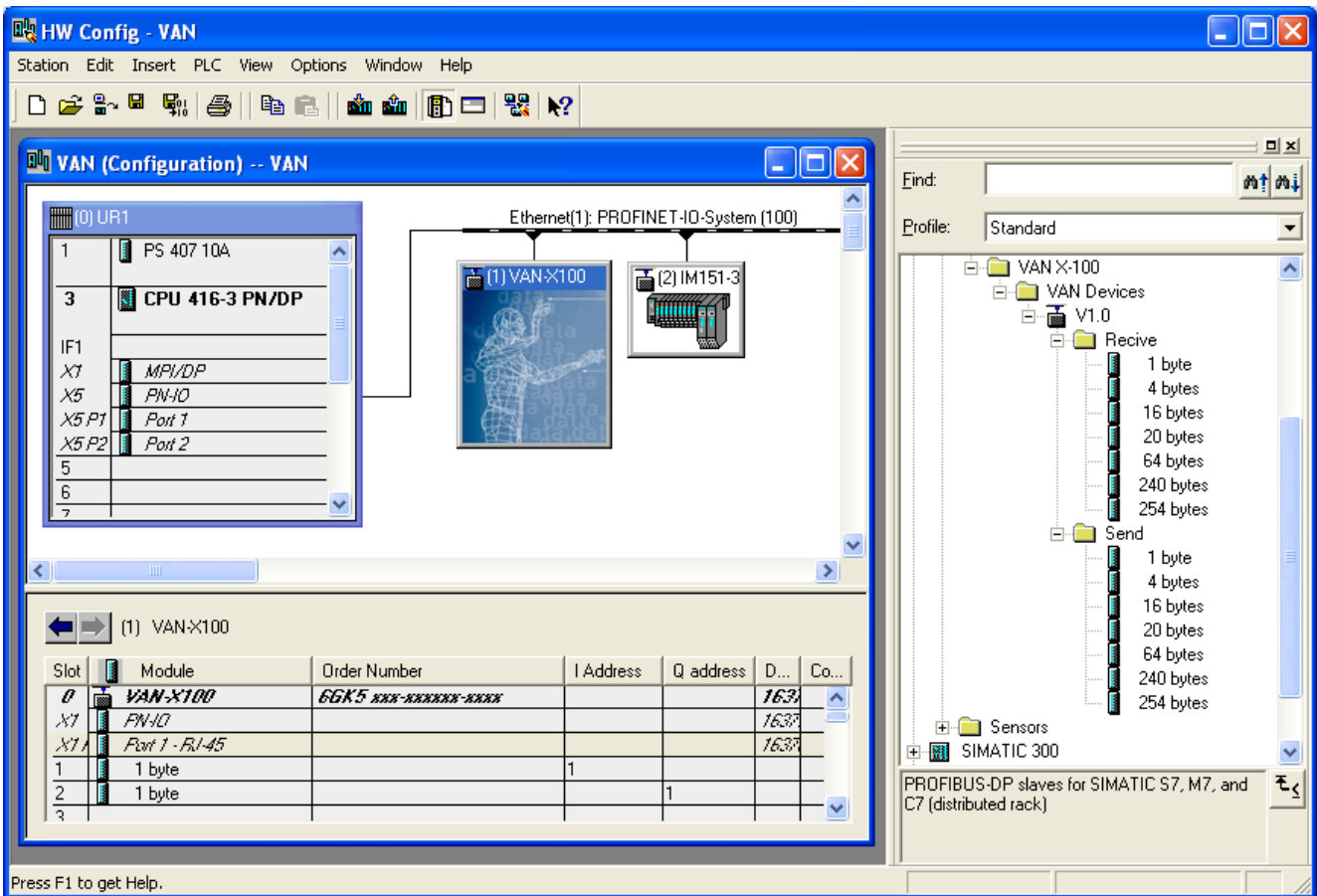


Figure 2: Adding a Profinet Device by Drag and Drop to IO-System

The local engineering system is reading the VAN Instance Model. The user can only use a VAN device, which is defined in the VAN Instance Model. This will be ensured through a consistency check of the local engineering system. Some definition also must be done for the VAN configuration. There exist attributes, which must be set for the VAN configuration. Figure 2 shows an instance of a VAN device. The ASEs are defined in the XMI file. Some VAN related parameters for the ASEs can be edited in STEP7.

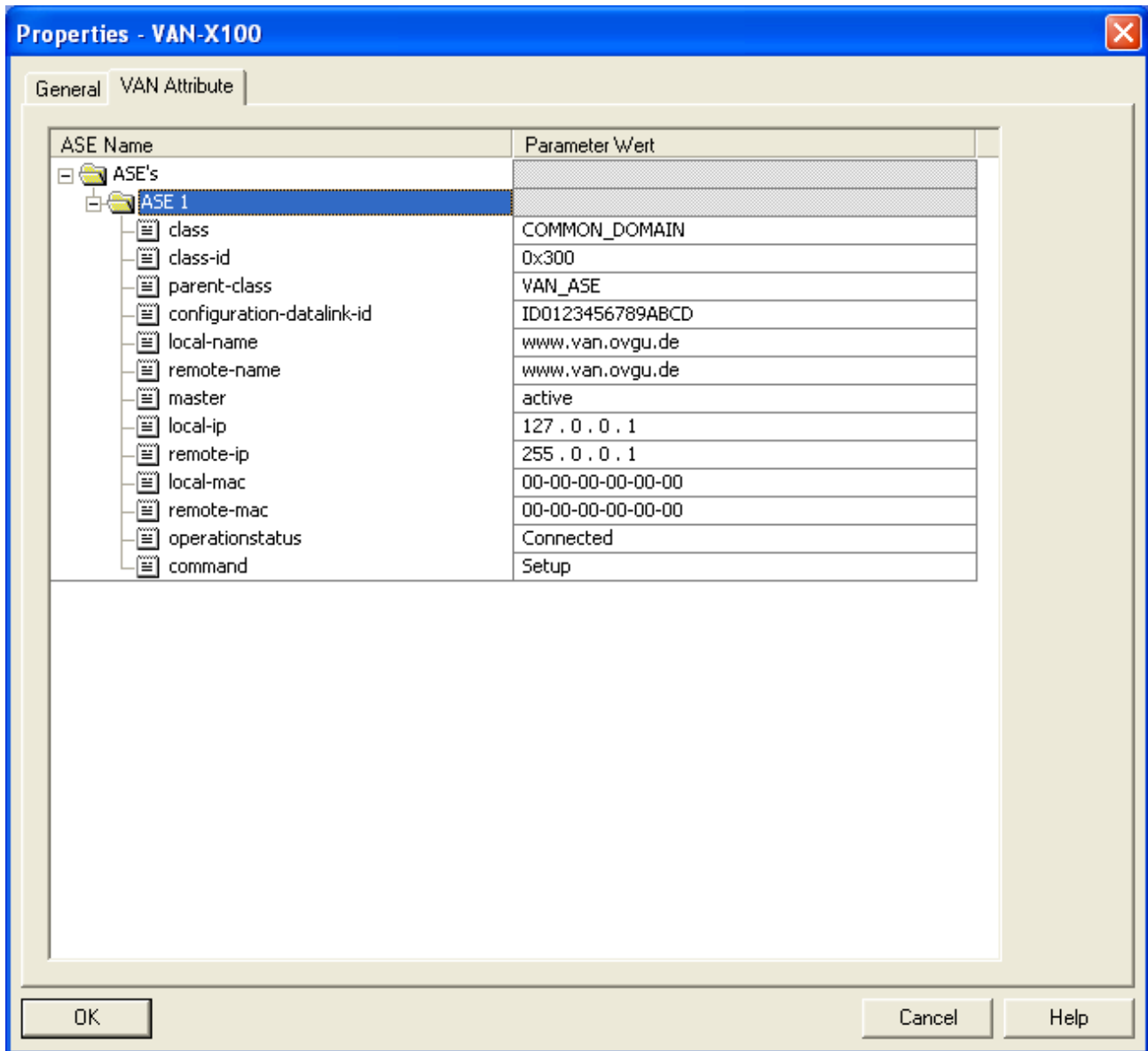


Figure 3: Editing Attributes for VAN Devices

Now the user can input the remote VAN device to import the available send transfermodules and request transfermodules to handle data from the local plant. It can be done by double-click the VAN device or by Drag and Drop: (hardware catalogue modules under "Receive" and "Send") There exist *sendtransfermodules* to send automation data in a VAN network segment and there exist *requesttransfermodules* to receive data over a VAN-Network.

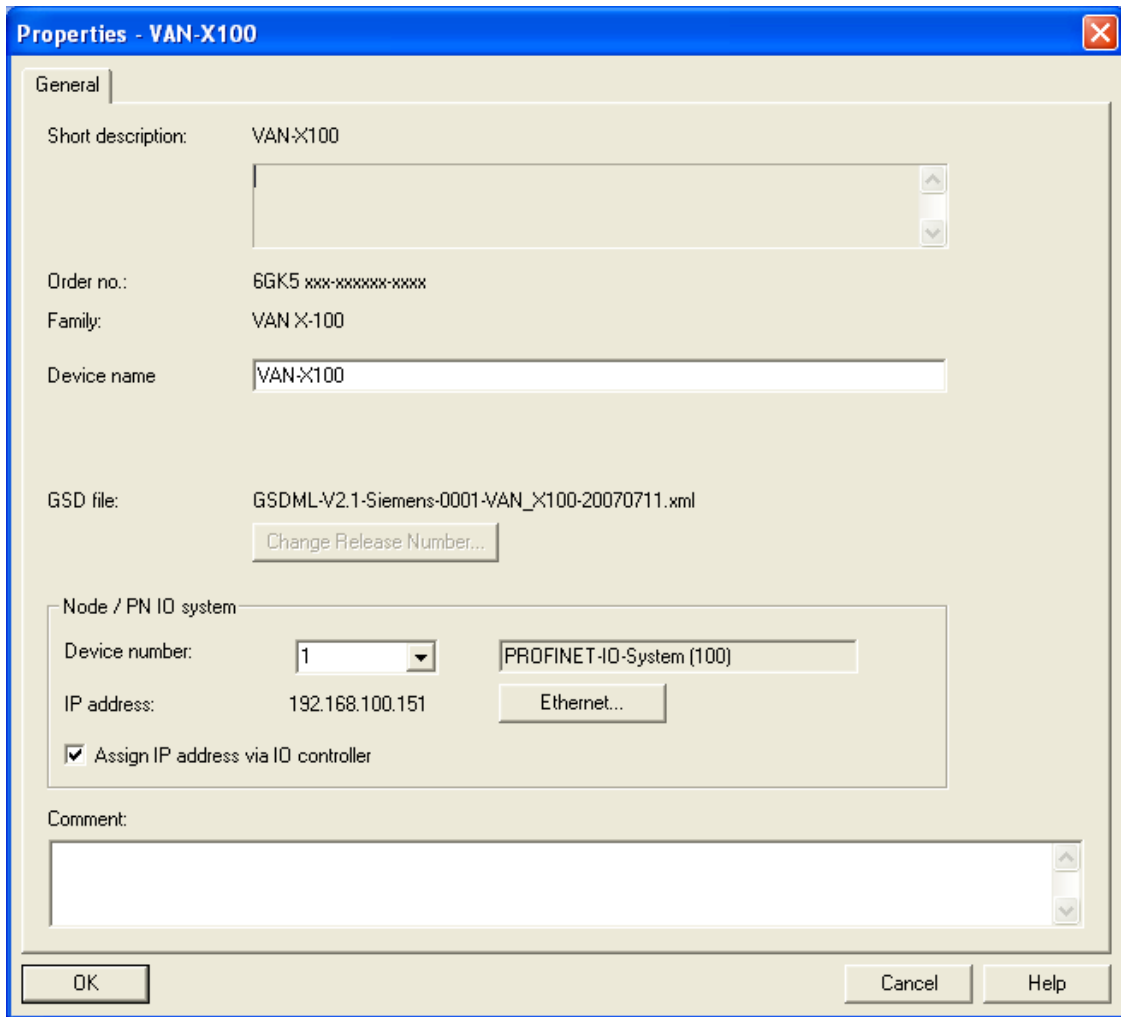


Figure 4: Access to the Profinet Part of a VAN Device

The VAN device can be configured by double-clicking the device. The according GSD will be shown and the user can edit the VAN device name and the Ethernet address at the local Profinet-IO device. In addition, also the user can define which of the 256 possible Profinet-IO device-numbers the device should get [IEC 61158].

2.1.1.3 Send and Request Transfermodules for VAN

The user can project different sizes of send and request transfermodules for the VAN device.

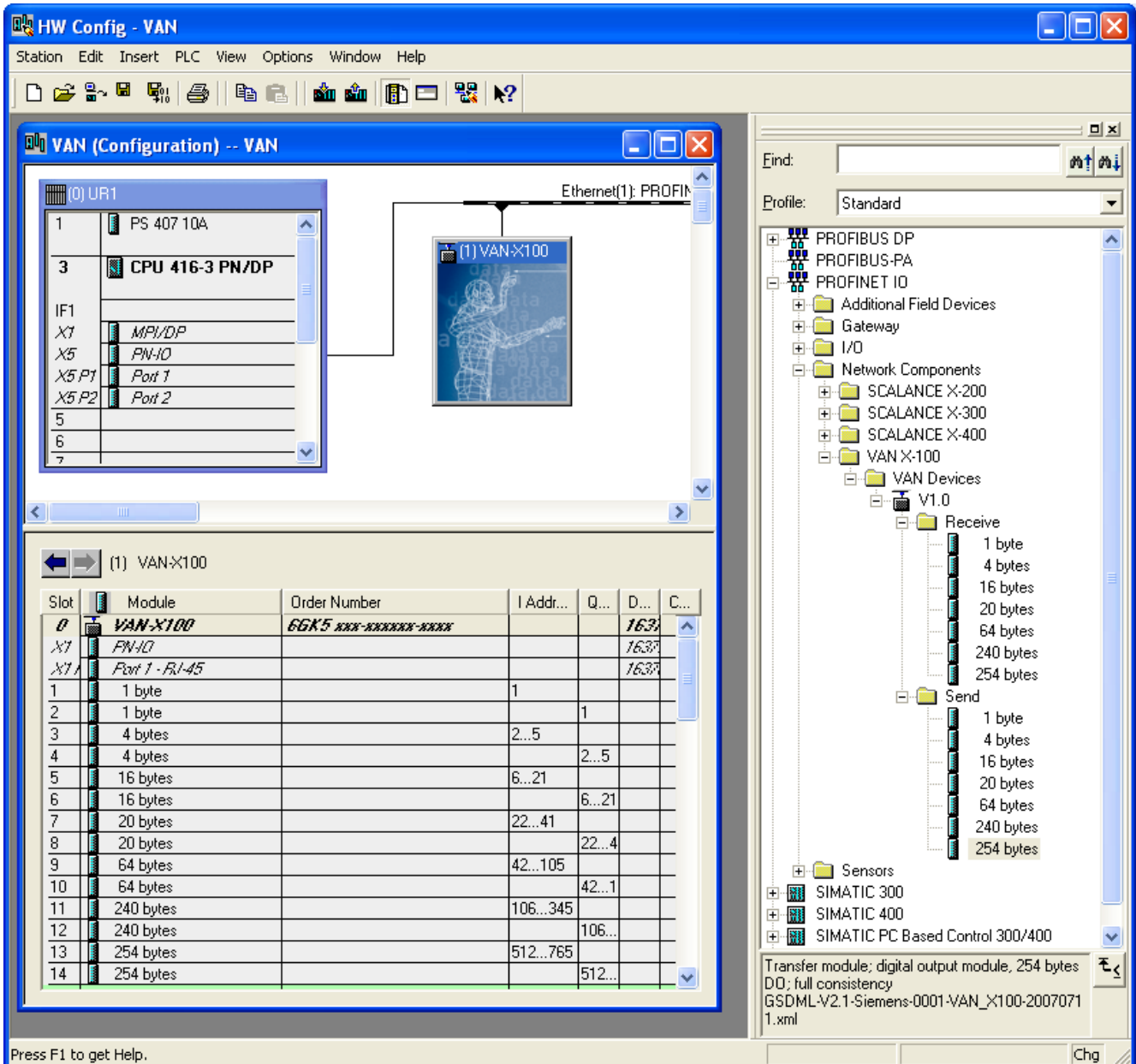


Figure 5: Send and Request Transfermodule

There exist different sizes of send and request transfermodules (see Figure 5). The sizes 1,2,4,16,20,64, 240, 254 bytes are planned. The engineer can define the send or request transfermodules, which can be remotely accessed by another VAN device. Each send or request transfermodule has an address, where to the VAN device should send or request the data defined by transferunits.

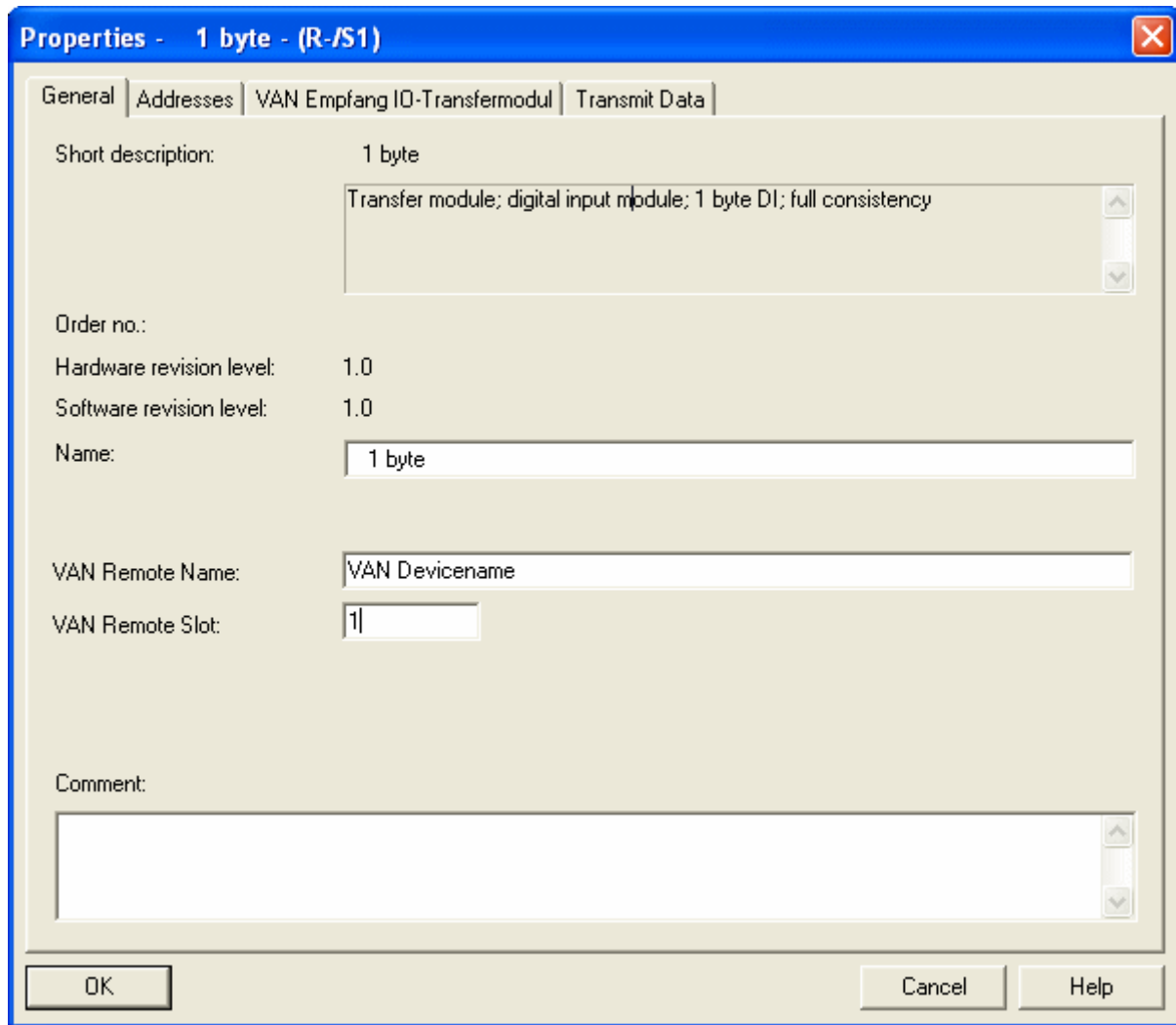


Figure 6: Send or Request Module with VAN Address

2.1.1.4 Send or Request Transferunits for VAN

Each transfermodule has transferunits, which define bit detail parts at a transfermodule. The send or request transferunits are mapped to any output or input ports of the modules of the remote device, i.e. the device with which the data is exchanged. Each request transfermodule has a corresponding send transfermodule at a remote device.

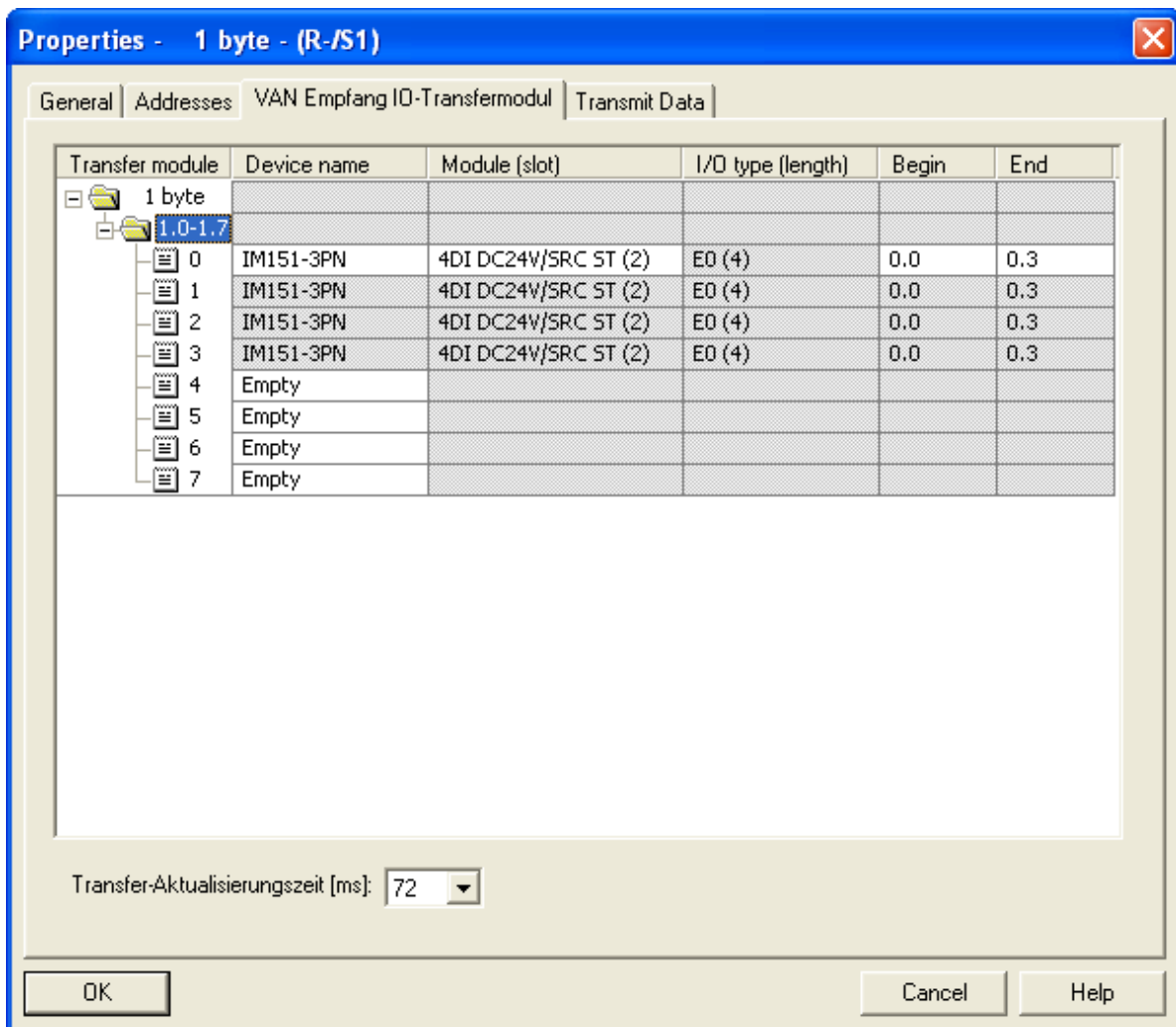


Figure 7: Send or Request Transferunits

The transfermodule shown in Figure 7 has 1 byte and starts at address 1. The engineer can choose any start bit to select from the local Profinet-I/O devices in the column "Device name". Thus the engineer can select any module of the defined device of the local Profinet-I/O in the column "Module". After that the engineer can position the local address of the module at the Profinet-I/O with the fields "Begin" and "End". The engineer can freely address any bit selection of the modules of the local devices, which are in the local project. This bit selection is called transferunit. A transferunit can be 1 bit or the address space of the whole transfermodule [IEC 61158].

2.1.2 TCI Call

With the Tool Calling Interface (TCI) specification [TCI] which was already described in [D08.4-1] it is possible to start any other program from the local engineering tool. The TCI is an interface, which bypasses information with a Temporary Parameter File (TPF file) in eXtensible Markup Language (XML) format. To enable the TCI interface with a context menu in local engineering tool some registry settings must be done. To enable the external file by a context menu in the local engineering tool some registry settings must be done. These settings are defined in the TCI specification and activate the TCI, e.g. in the local engineering tool like Step7.

The number {11111111-1111-1111-1111-111111111111} in this example is a client-server identification for the external device configuration program, i.e. the Device Tool (see chapter 5). The number is only an example, but the format should be used. The code 0x002A represents the vendor-ID given by Profinet-I/O organisation. The code 0x0FFF represents the product-ID which is defined by the vendor.

Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\PROFIBUS International]

[HKEY_LOCAL_MACHINE\SOFTWARE\PROFIBUS International\TCI]

[HKEY_LOCAL_MACHINE\SOFTWARE\PROFIBUS International\TCI\PN-Devices]

[HKEY_LOCAL_MACHINE\SOFTWARE\PROFIBUS International\TCI\PN-Devices\0x002A-0x0FFF]

[HKEY_LOCAL_MACHINE\SOFTWARE\PROFIBUS International\TCI\PN-Devices\0x002A-0x0FFF\{11111111-1111-1111-1111-111111111111}]

[HKEY_LOCAL_MACHINE\SOFTWARE\PROFIBUS International\TCI\Tools]

[HKEY_LOCAL_MACHINE\SOFTWARE\PROFIBUS International\TCI\Tools\{11111111-1111-1111-1111-111111111111}]

"AppPath"="C:\Program Files\Siemens\Step7\I57VAN\VAN_ECD.exe"

"PID-File"="C:\Program Files\Siemens\Step7\I57VAN\VAN-PID-Profinet.xml"

In this example the TCI is adding a context menu, which is defined in the Program Interface Description file (PID file), "VAN-PID-Profinet.xml", and if the user is using the context menu, the defined program "VAN_ECD.exe" is started and takes the TPF file as parameter, which the local engineering tool is generating TCI conform.

2.1.2.1 Context Menu through TCI

To make these settings visible in a local engineering tool, there must be defined a PID file, where the context menus for visualization are defined. Details are given in chapter 5.1

In the PID file of the example VAN X-100 (VAN-PID-Profinet.xml) the menu is defined, the PID file example you will find in chapter 5.1.1 Each entry of the context menu in the local engineering tool can start a function of the Device Tool.

2.1.2.2 Reading VAN Device Description

The Device Tool, e.g. the AutomationXplorer+ with a Field Device Tool container (FDT container) for VAN, is started by TCI and the TPF file, which the local engineering tool generates, is passed to the Device Tool. In the TPF file additional parameters will be found and will also be provided by TCI to the Device Tool through attribute value pairs. The FDT container will use the VAN device description to identify the ASEs of the VAN device. Additional parameters are given by the VAN Device Device Type Manager (DTM)(see chapter 6.2). Those engineering parameters must be stored for the VAN device configuration.

2.2 Stand-Alone Concept

The following description explains an approach to engineer a VAN system with the help of the stand-alone concept described in [D08.4-1].

The stand-alone concept differs from the approach of the integrated concept, because it is independent from vendors and a free fitted design is usable. Additionally, it follows a hierarchical approach (see Figure 8), starting with the generation of a VAN domain respectively sub domain. From there on it goes down to the devices and ASEs. The well-structured concept allows the top-down approach – which is described below – as well as the bottom-up approach, if a VAN domain exists. Both directions have their benefits, strongly dependent on the use case and the engineer. For using both approaches, a common repository like shown in the Figure 8 is indispensable. It stores all VAN dependent information and provides them for the authorised users.

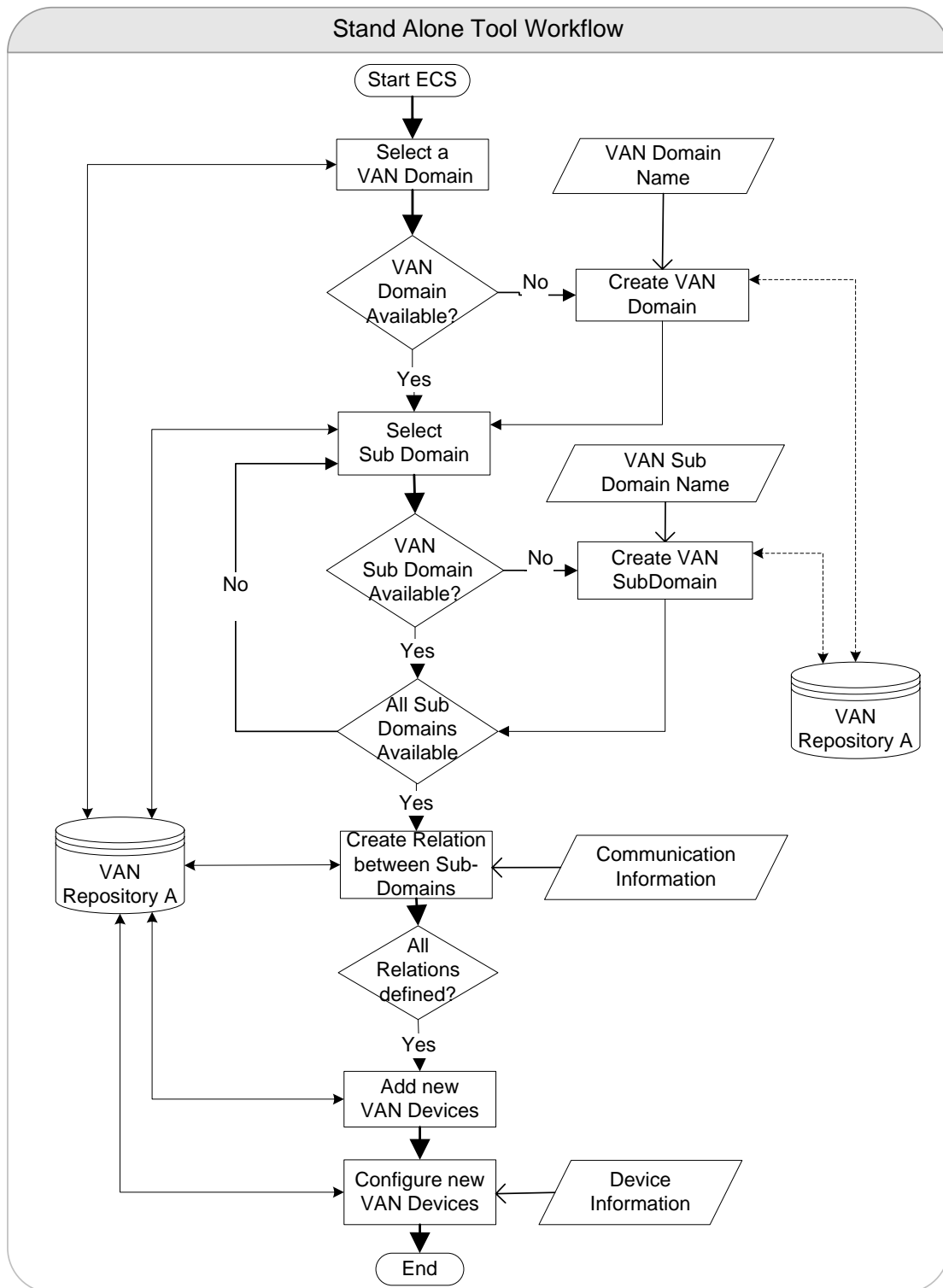


Figure 8: Approach for a Stand-Alone Concept

2.2.1 Create and Select a VAN Domain

The starting point of VAN engineering is the creation of a VAN domain. This means choosing a name and storing it - for example in a common server for centralised access, which serves as the necessary repository for a monolithic approach. It has to be accessible to the engineer from any place at any time if he has the required access rights. Because of its importance for VAN, the right to create a VAN domain has to be restricted (see dashed arrows in Figure 8). If a user decides to work in the lower levels of hierarchy, for instance to add a VAN device, then he nevertheless must access a VAN

domain (see chapter 1.2.1 of [D02.2-1]). For further work, one of the existing domains has to be selected. Additionally, it is important to mention that the creation of a VAN domain should be strongly restricted by the used tool or by the access right list managed by the repository. A proposal how the access can be organised is explained in chapter 6.1.3 in this deliverable.

2.2.2 Create/Select a VAN Sub Domain

The next step in engineering is the creation of a VAN sub domain. If only a small system has to be engineered with no distributed locations or different vendors the designed VAN domain is coeval a sub domain. Due to the hierarchical structure of the VAN system, every VAN device belongs to a VAN sub domain and every sub domain is part of a VAN domain or another VAN sub domain. This means that a VAN device is always placed in a sub domain and the sub domains belongs always to a VAN domain, if more than one domain exist.

The sub domains will be stored in the common repository as well. Creating a VAN sub domain is a repeating task. Therefore, the access rights have to be less restricted. It should be possible to not restrict this action at all. However, in the same context, this leads to necessary maintenance cycles

2.2.3 Build up Interconnections between VAN Sub Domains

Two kinds of interconnection classes for VAN sub domains have to be distinguished. Firstly, the logical connection – which itself should be separated into the two views of a sub domain and a end device layer – and, secondly, the physical connection – that means the communication path between two named devices. The physical connection is not relevant for the engineering process at this configuration level. At the sub domain layer, the logical connection describes the relation between sub domains, whereas it describes at the device level an end-to-end relation between two VAN devices. The latter relation will be created automatically if the VAN device has information about the target VAN device.

If a VAN network is engineered with the top down approach, interconnections between VAN sub domains will be defined, before VAN devices are assigned. That means the logical connection is specified (see Figure 9) without any information about the used VAN access points (VAN-AP; see chapter 1.2.2.3 of [D02.2-1]). At this point of engineering “Quality of Service” (QoS) requirements will be defined. The information comes either from the later used devices with their specific requirements to fulfil their automation tasks, from the customer or from the used infrastructure. The last two mentioned points are part of the VAN Instance Model, which is explained in chapter 3.2

If later on the sub domain interconnection endpoints – the VAN-APs - are assigned and configured, possible conflicts between the QoS requirements and the network abilities of the devices will be signalled via messages and colour codes. Figure 9 shows how the logical connection is established, if VAN-APs are already defined within each sub domain. While building up this relation, a data exchange of necessary information is made available. The kind of information could be, for example, all the names of VAN devices that are accessible from another sub domain.

2.2.4 Add New Device

For adding new VAN devices, planning prerequisites – like a VAN domain and sub domain - must be fulfilled. The VAN device represents an object with a required functionality for the automation task in a VAN domain and is placed within the sub domain view under consideration that more than on domain exists. Each device has a default set of stored information - the so-called ASEs. If a new device is added, the type of this device has to be selected by the user. Some ASEs are filled out with a default data set, provided by the device vendor, in a VAN device description file, which is explained in chapter 4. The classification of devices with their specified functionalities is described in detail in [D02.3-1].

After adding a new device, the engineer has to name this device. The name should follow the used naming convention. The structure of the name is already fixed by the corresponding VAN domain and sub domain. Just the preamble of the name has to be chosen.

2.2.5 Configuring new VAN Devices

The configuration of the added VAN devices is also a stepwise activity. In context with the introduced VAN Engineering concept the following tasks are part of the VAN Engineering Client for automation Device (VAN-ECD) [D08.2-1] while the steps above are part of the VAN Engineering Client for automation System (VAN-ECS) [D08.2-1].

First, the newly added VAN device has to be connected to other available devices, independent from their location. For this purpose, information about connectable devices must be present which are coming mainly from the VAN Instance Model explained in chapter 3.2. Following, several possible ways for obtaining this information will be explained.

The *concept 1* in the Figure 9 assumes that a relation between sub domains exists. If VAN-AD3 wants to build up a communication relation to VAN-AD4, the engineer has at first to establish a connection to a VAN-AP, which is the interface to other sub domains. The VAN-APs should have the functionality – within the engineering tool - to store data (e.g. names) of all connected devices within their sub domain and to exchange this information with all connected VAN-APs. This can be realised manually by the engineer or automatically. If the VAN-APs keep this information accessible, a connected VAN-AD is able to retrieve it and to pick up the name of its target VAN device within another domain. However, only if the logical connection between the VAN sub domains is available, the logical connections between VAN devices can also be designed. This data exchange between two sub domains is useful only during the engineering process - that can also be a distributed process for configure the device. Finally, the physical device must not support this data exchange

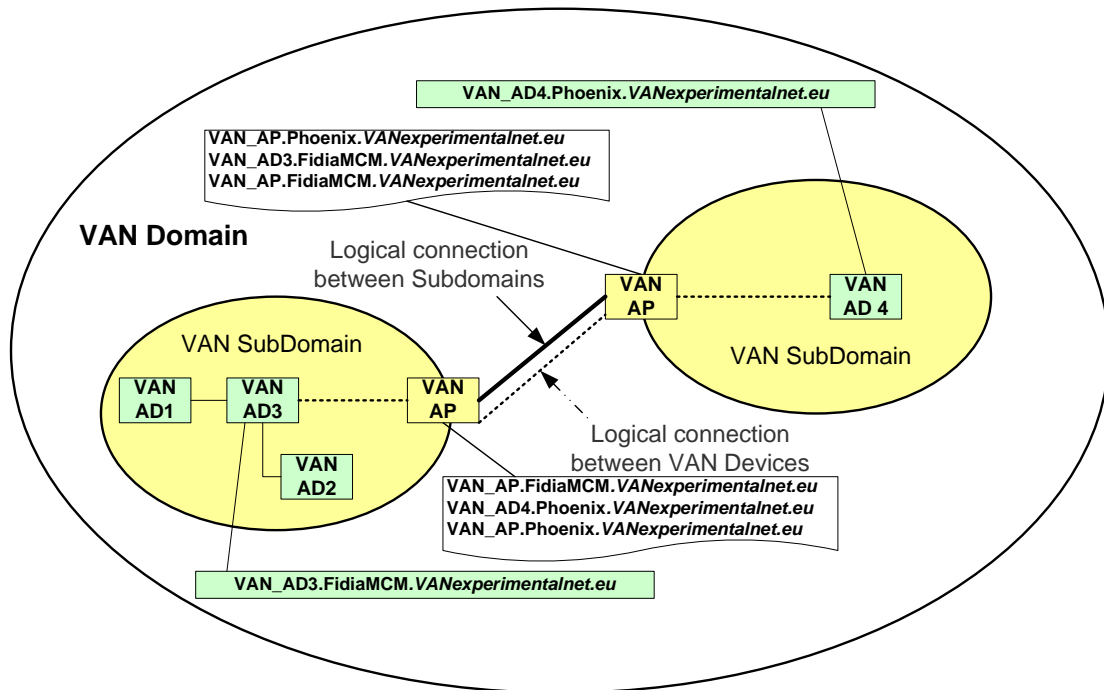


Figure 9: Engineering of Communication Connections - Concept 1

The *concept 2* in Figure 10 represents a different approach. Here the document objects *Sub domain A*, *Sub domain B* and *Device AD4* are introduced to represent the interfaces to other sub domains and devices within a VAN domain – coeval with the used VAN-AP in chapter above. After adding a new VAN device, the engineer requests information about the target address of the communication. After acquiring this information, for example from the responsible engineer of the connected sub domain, he is able to create the VAN virtual container device. This virtual container device serves as related virtual representation to the physical VAN device in another VAN sub domain.

It stores the target device information for establishing a logical connection between the two VAN devices. An internal mechanism should validate if the generated device is reachable via the

connected sub domain. If there is no relation between sub domains, this virtual container device cannot be generated.

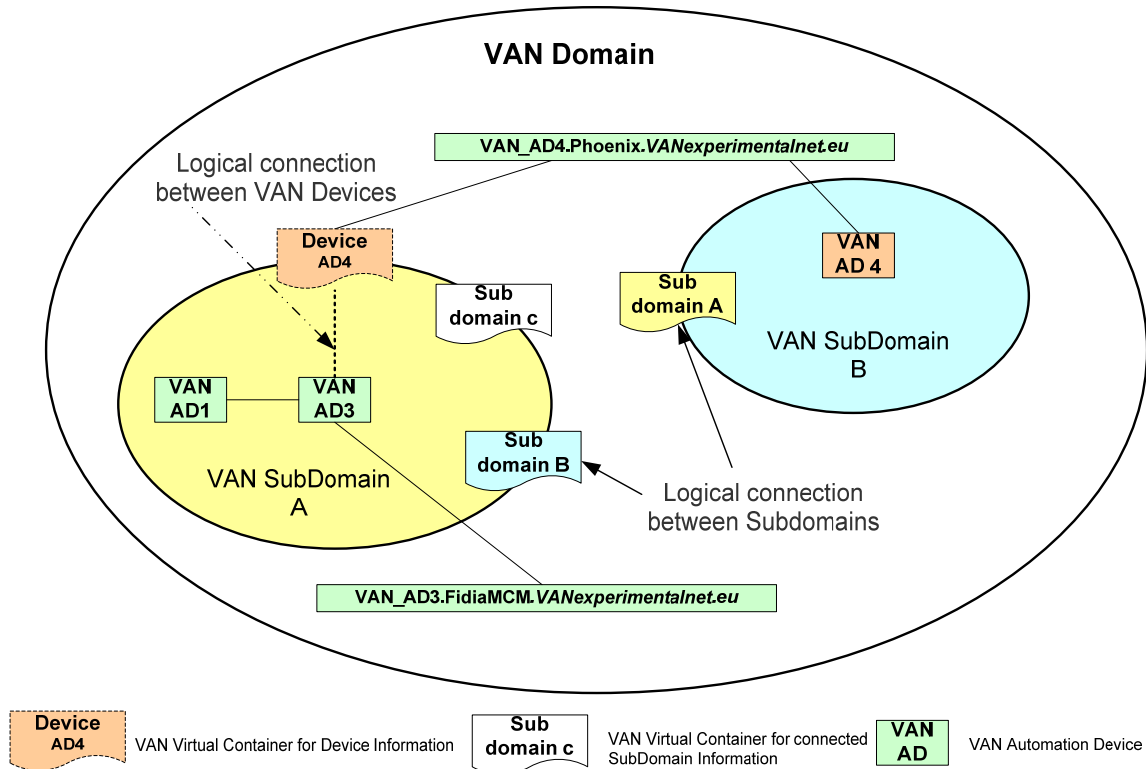


Figure 10: Engineering of Communication Connections – Concept 2

Both above mentioned concepts are possible ways to engineer communication relations independent from their location and the used communication technology. In both case a common repository to manage the data is necessary.

As already described, the configuration is a stepwise approach. The second step is the configuration of the devices in order to establish a physical connection and to provide other functionalities. For this purpose, the ASEs are used. The parameter sets configured within the different ASEs can be read and written from/to the device. The following paragraphs describe the approach in principle.

2.2.5.1 Read ASEs from Device

To read an ASE, the first step is to select a VAN device. If a VAN device is selected, all available ASEs are displayed. With help of a trigger - e.g. a read button - the attributes, which contain default parameters provided by the vendor, are read out from the device and they are shown in the used tool.

The process to acquire the ASE from the devices is based on web services. This means, that both web service provider and a web service consumer have to be available. The provider, in this case a web service server, is implemented in the VAN device. Previously to the engineering process the web services are defined by means of the Web Service Description Language (WSDL) file. One predefined service grants access to one ASE. That means each defined ASE has one service to set the parameter and on service to get the parameter. The engineering tool uses this web service server implemented into the VAN devices to request the needed information. It is essential to know the Uniform Resource Identifier (URI) of the web service server within the adequate VAN device. This required data should be provided by the centralised repository when the user selects the respective device in the sub domain.

2.2.5.2 Demonstration Implementation

Parallel to the development of this stand-alone concept, a prototype tool is developed where parts of this concept are implemented for evaluation.

Figure 11 shows a part of the graphical user interface (currently implemented as prototype) where the access to the web service is realised by using the target device name. The mentioned prototype engineering tool accesses to the web service by using IP addresses at the moment. This means that the address of the VAN device web service is not provided by a domain server, but by the user. In future the IP address will be replaced by the unique name of the VAN device – like defined in the VAN naming convention. If the VAN device name is well known the name can be used by the tool to establish a connection to the web service. The prototype provides this functionality by a form which has to be filled with the target device name. After filling the form, the engineering tool can connect to the web service. After the selection of the device configuration of a service and push the “get” button the appropriate ASE with all its available attributes is displayed.

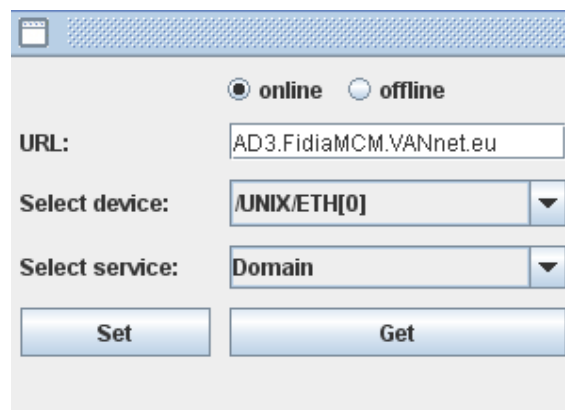


Figure 11: Control Elements of the GUI

If the engineer has the adequate access right, he is able to modify the shown parameters after getting the ASE information. The modification can be also done in an offline mode of the engineering tool.

2.2.5.3 Write ASEs to the Device

Every ASE parameter is loaded with a default value or with empty fields where no default values are set to show the content. The default value parameter set is typically delivered by the vendor of VAN devices in a VAN device description file (see chapter 4). For the first prototype implementation, form fields, shown in the tool in Figure 12, are used. Therefore, available values can be easily changed and empty fields can be filled.

The screenshot shows a window titled "/UNIX/ETH[0] - DomainASE". The window contains a form with the following fields:

Category	Field Name	Value
basic	class-ID	0x0300
	parent-class	VAN ASE
configuration	link-ID	ID0123456789ABCD
	local-name	http://tempuri.org
	remote-name	www.van.ovgu.de
	master	passive
	local-IP	141.44.140.133
	remote-IP	141.44.140.111
	local-MAC	00-00-00-00-00-00
	remote-MAC	FF-FF-FF-FF-FF-FF
operation	status	Disconnected
	command	Setup
	parameter	00-00-00-00-00-00

Figure 12: Screenshot of ASE View

After modifying the parameters of the ASE, a write functionality, that sends the new configuration set back to the device, is necessary. The sending is the reverse process of reading ASE. The roles of the web service server for the VAN device and web service client at the engineering tool side are the same, but the request contains the edited ASE and the respond confirms the successful operation.

2.3 Storing of a VAN Project

Every VAN Project is available as a set of data. This data are stored in XML files. As such, they are connected via references. For instance, the ASE files are build from parts. There is one file, the ASE, that contains the basic information and part of this may be the domain ASE. That ASE is referenced by its name and stored in another file. Another example, a sub domain and all its basic data are stored in one XML file. All connection information as part of the sub domain are files itself. They are clearly identified by there name and as such referenced in the sub domain file.

A tool functionality allows to import or export whole VAN projects via XML files. This functionality has to be strong protected against unauthorised access. These authentication procedures are one of the future tasks that have to be realised but out of scope for the engineering process.

3 VAN Information Model

3.1 Update of the Information Model

The VAN Information Model was originally developed in deliverable D08.3-1 as part of the VAN Engineering Tool architecture. The Unified Modelling Language (UML) based information model considers all aspects concerning the design of a VAN network beginning with the description of network infrastructure until the description of the Application Service Elements (ASEs), which are used for VAN communication. During the review process of the VAN Information Model in task T8.4, new and enhanced ASEs were added. An aspect that should be mentioned is the new relation named *Domain_Affiliation* between the class Network Segment and VAN domain within the network view shown in Figure 13. The reason why this new relation was added is that each VAN Network Segment requires a name derived from the VAN domain name. The multiplicity is free scalable on both sides. Additionally the Zigbee technology as VAN technology layer was also added. Nevertheless, in future reviews maybe some more new classes and relations have to be added because of the continuously development process

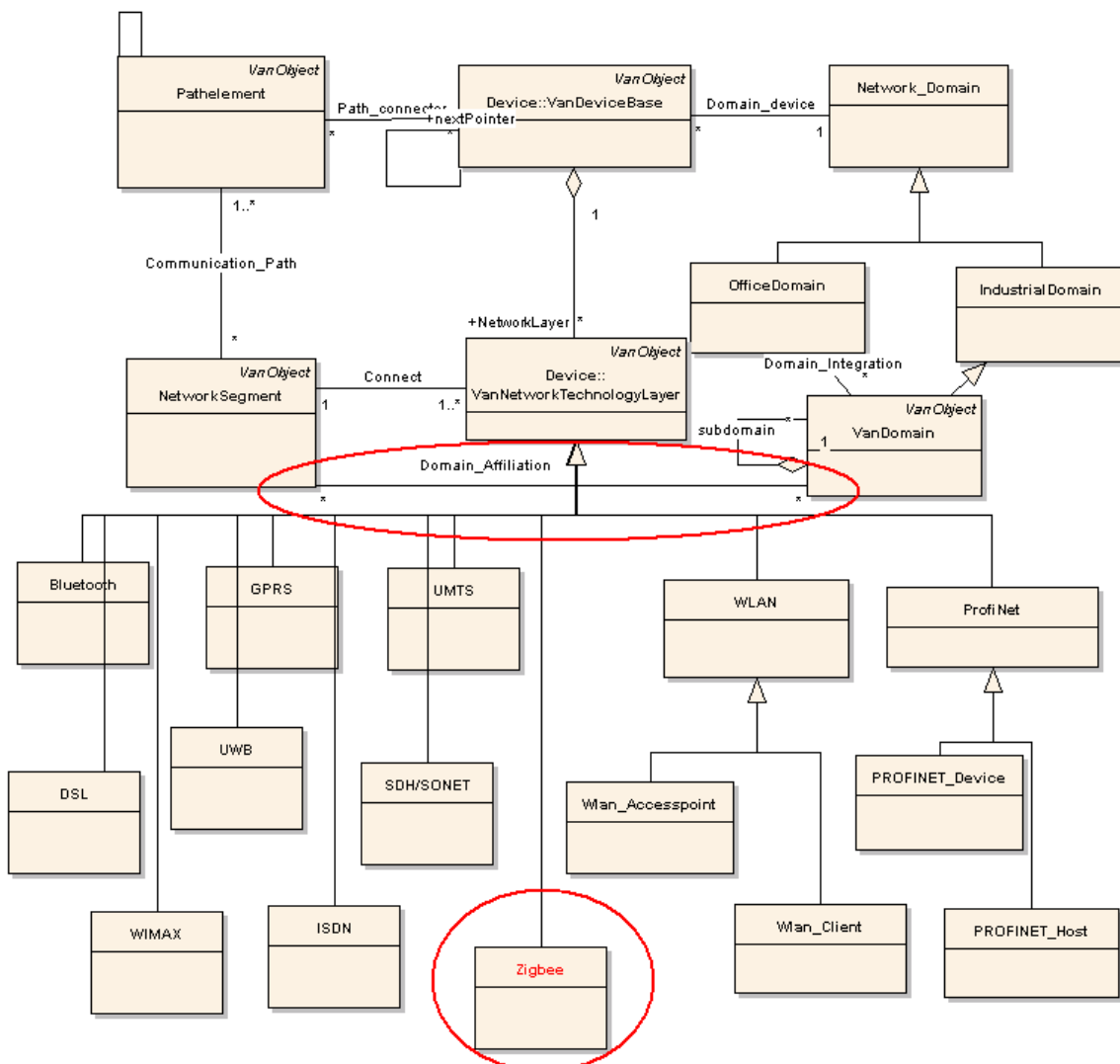


Figure 13: VAN Network View

3.2 VAN Instance Model

The next sub chapters describe how the VAN Information Model can be used for engineering a specific scenario. For this, an instance derived from the complete information model has to be created. The benefit of this approach, instead of engineering a use case independent, is that all required objects with their relations are available for the engineer in the information model.

3.2.1 Characteristics of the Use Case

Based on the instantiation of the “Reduced Example of a Factory Automation Topology” of deliverable D08.4-1 (see Figure 14) the following paragraph describes the developed instance model derived from this use case. The instance is created by using the generic VAN Information Model, based on the engineering workflow approach developed in deliverable D08.2-1.

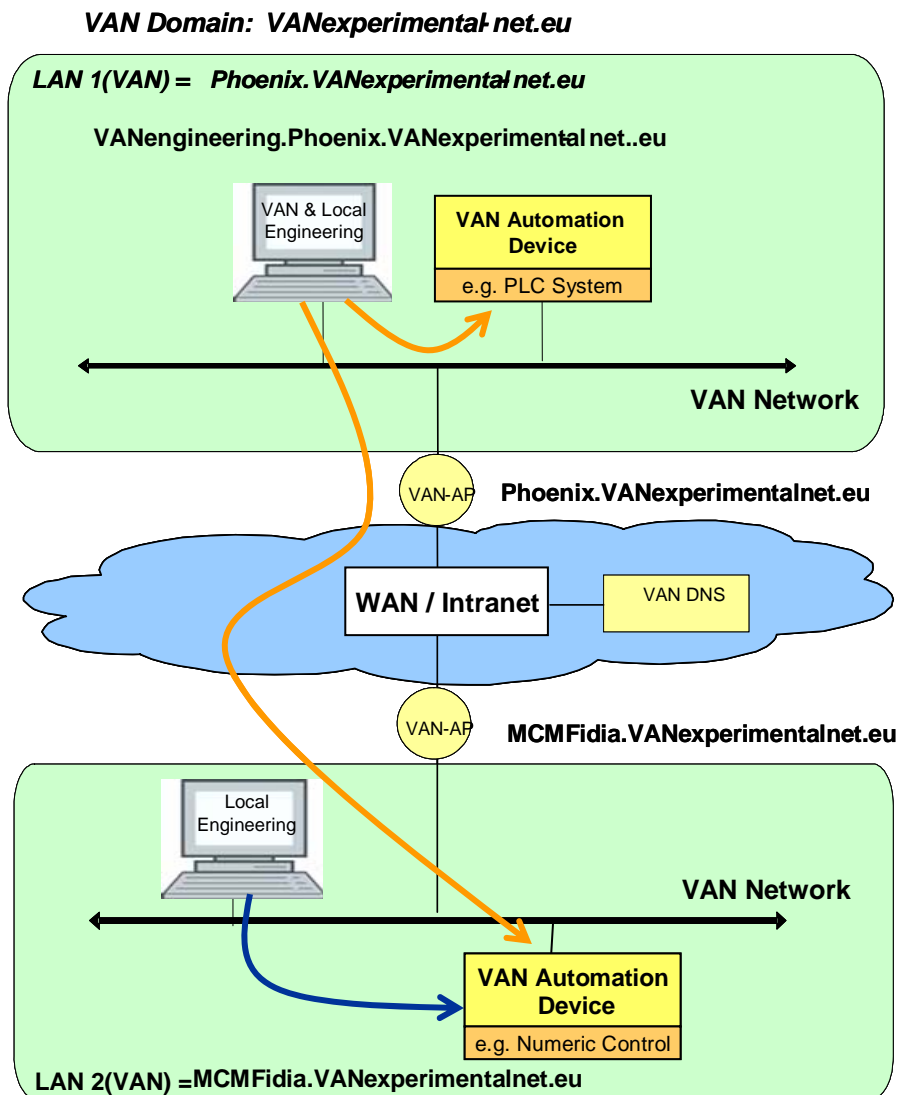


Figure 14: Use Case Scenario

Before starting the creation of the instance of this use case, requirements concerning the topology and used devices must be specified. For this case, the following attributes are defined:

The VAN domain name is:

VANexperimentalnet.eu

LAN 1

The physical network uses the 10/100Mbit/s-Ethernet (IEEE 802.3) technology.

LAN 1 is a VAN network segment covered by VAN sub domain named:

Phoenix.VANexperimentalnet.eu

LAN 2

The physical network uses the 10/100Mbit/s-Ethernet (IEEE 802.3).

LAN 2 is a VAN sub domain named: ***FidiaMCM.VANexperimentalnet.eu***

VAN-AP within LAN 1

The VAN-AP is used for communication between VAN-ADs in MCM/Fidia LAN and Phoenix LAN. The used name is for LAN1 is: ***VAN_AP.FidiaMCM.VANexperimentalnet.eu***

VAN-AP Within LAN 2

The VAN-AP is used for communication between VAN-ADs in Phoenix LAN and MCM/Fidia LAN. The used name is for LAN2 is: ***VAN_AP.Phoenix.VANexperimentalnet.eu***

Both VAN-APs follow the specification made in deliverable D02.3-1 for VAN-AP profile with conformance class A. The conformance class A defines basic subsets detailed in Table 1 [D02.3-1]

Major layer	Feature/Object	Description	Class A	Remarks
VAN Network Layer	VAN Heterogeneous Network Technology Adaptation Layer	See D02.2-1 chapter 2.4.10	m	The implementation of these Interface should consist of typical driver operations like open(), close(),read(), write() and iocontrol(). The implementation of the interface is local matter.
	IEEE 802.3	Access to standard Ethernet network.	m	The access over IEEE 802.3 to VAN communication environment
	Fie lower layer	Standard Profinet Driver	m	It is the common access to the VAN communication environment.
VAN Communication Stack	VAN Runtime Object Dispatcher	Dispatches automation objects conveyed via the runtime object tunnel.	m	Implementation is local matter and should be optimized to fit the vendor specific upper layer. Examples of using in combination with VAN runtime object tunnel see D02.2-2 Chapter 9.
	VAN Runtime Object Tunnel	The tunnel is an interface to the application layer.	m	The tunnel is divided into different levels one see Application programming interface (API) description D02.2. Example of using in combination with VAN runtime object tunnel see D02.2-2 Chapter 9.
	Web service over Virtual Automation Networks	Provides the WS of the VAN device	m	The name based access via web service to the devices is one key concept of VAN.

Major layer	Feature/Object	Description	Class A	Remarks
	ACL	Access Control Layer assures the authorised access to objects in the VAN device.	m	The functionality is topic of WP 6.
	VAN Common Communication Application Service Elements	For all ASE see API specification in D02.2.-2	m	A minimal subset of ASE is need for each VAN device.
	VAN Domain		m	
	VAN Dev Config		m	
	VAN Access Management		m	
	VAN Switching		m	
	VAN Peer-to-Peer		m	
	VAN Routing		m	
	VAN Time		m	
	TCP/IP Stack		m	Defines TCP/IP Stack for the Web Service Access to the Device.
	HTTP	HTTP Protocol as carrier for SOAP	m	
	DNS		m	
	DHCP		m	
	ICMP v4/v6		m	
	ARP		m	
	IPv4/v6		m	
	TCP		m	
	UDP		m	
	VAN Device and Network Management		m	
Time Sync	Realizes the time synchronisation between VAN Devices: - Within one LAN - Over InterLAN - or WAN	m	The necessary attributes an the specification of the messages is done in D04.3	
Application Layer	VAN Automation Application Process		f	
	VAN System Management Application Process	Controls all parts of VAN Communication Stack and Network Technology Layer	m	
	VAN Security Application Process		o	The functionality is topic of WP 6
	Parameter Config.	Is a kind of data bases for all defined parameters.	m	Discussion in Tech PCC which kind of storing mechanism will be provided by VAN devices.
	VAN PnP Application Process	Is used for automatic configuration and identification of connected VAN- Devices.	m	
	VAN Proxy Application Process		f	A VAN-AD does not provide any Proxy Function for other Devices.
	Web Server		m	

Table 1: VAN-AP Profile

VAN Engineering

The VAN Engineering Tool is not permanently connected to the network. Anyway, when it goes online it must also have a name depending on the connected LAN. For this use case, the VAN Engineering Tool is connected with LAN 1 and it is named: **VANengineering.Phoenix.VANexperimentalnet.eu**

VAN Automation Devices

The automation devices in this use case follow the specification made in deliverable D02.3-1 for VAN-AD profile with conformance class A. Conformance class A defines basic subsets detailed in Table 2 [D02.3-1].

Major layer	Feature/ Object	Description / Access to	Class A
VAN Network Layer	VAN Heterogeneous Network Technologie Adaption Layer	See D02.2-1 chapter 2.4.10	m
	IEEE 802.3	Access to Standard Ethernet Network.	m
	Profinet lower layer	Standard Profinet Driver	m
VAN Communication Stack	VAN Runtime Object Dispatcher	Dispatches automation objects conveyed via the runtime object tunnel.	m
	VAN Runtime Object Tunnel	The tunnel is an interface to the application layer.	m
	Web service over Virtual Automation Networks	Provides the WS of the VAN device	m
	ACL	Access Control Layer assures the authorised access to objects in the VAN device.	m
	VAN Common Communication Application Service Elements	For all ASE see API specification in D02.2.-2	m
	VAN Domain		m
	VAN Dev Config		m
	TCP/IP Stack	Normal TCP/IP stack.	m
	HTTP	HTTP Protocol as carrier for SOAP	m
	DNS	Domain Name System - stores information associated with domain names in a distributed database on networks. Because VAN is based on domain names, it should be mandatory for all devices.	m
	DHCP	Dynamic Host Configuration Protocol – client/server protocol based on UDP which takes care about assigning IP address based on MAC address.	m
	ICMP v4/v6	ICMP is an integral part of IP and is implemented by each device with an IP stack. All devices with web services rely on this	m
	ARP	ARP is a core protocol of the Internet protocol suite and is typically used for error responses in IP datagrams or for diagnostic and routing purposes. Inseparable part of IP stack.	m
	IPv4/v6	Internet Protocol stack. All devices with web services rely on this because are based on HTTP and HTTP is based on TCP/IP.	m
	TCP	Transmission Control Protocol - one of the core protocols of the Internet protocol suite. The protocol guarantees reliable and in-order delivery of data from sender to receiver. All devices with web services rely on this.	m
UDP	User Datagram Protocol - UDP does not provide the reliability and ordering that TCP does. Compared to TCP, UDP is required for broadcast (send to all on local network) and multicast (send to all subscribers).	r	

Major layer	Feature/ Object	Description / Access to	Class A
Application Layer	VAN Automation Application Process	Fulfils the automation function of the VAN-AD.	m
	VAN System Management Application Process	Controls all parts of VAN Communication Stack and Network Technology Layer	m
	Parameter Config.	Is a kind of data bases for all defined parameters.	m
	Web Server		m

Table 2: VAN-AD Class A Subset

The VAN-AD within LAN 1 is named: ***VAN_AD1.Phoenix.VANexperimentalnet.eu***

The VAN-AD within LAN 2 is named: ***VAN_AD2.FidiaMCM.VANexperimentalnet.eu***

3.2.2 Design of the VAN Instance Model

The instance model, based on UML, is segmented into different stages. Within the first described approach in earlier deliverables, the idea was to put any needed information into the instance model. For an easier handling, the following description will use a separation between VAN Instance Model information and ASE information.

The first stage – a task for the VAN-ECS described in deliverable D08.2-1 (see chapter 1.1.3 of [D08.2-1]) – is the design and creation of all objects to realise the dimensioning of the topology and its relations (orange triangle in Figure 15). The second stage describes in detail the configuration of the VAN devices created in the first stage (yellow triangles in Figure 15).

The instance model file – provided as XML standard 1.1 - must be available for all parties to realise the defined functionality. The detailed device description, also saved as XML file and named “VAN device description instance file” (VAN-DD instance file; see chapter 4.3), can be engineered later on with a VAN-ECD. As result from the dimensioning of the VAN domain with a VAN-ECS, there exist close couplings between the devices from different sub domains, which have a relation with each other. The VAN Instance Model itself represents only this relation and not the detailed device configuration.

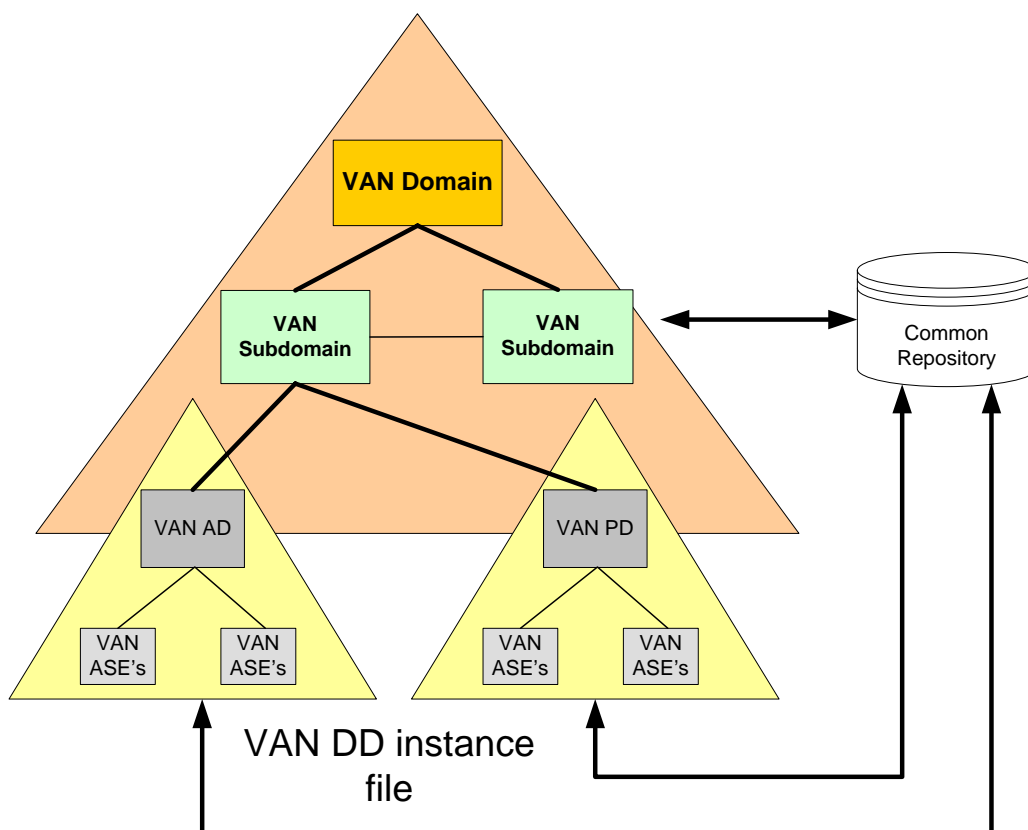


Figure 15: Instance Files

With the use case shown in Figure 14 a scenario to proof the concepts of VAN engineering is available. This test scenario allows the engineering of two devices within each VAN sub domain connected via public networks. Figure 16 visualises the use case scenario as an UML instance derived from the available VAN Information Model. Figure 16 shows on the left side the device objects, which represent devices within MCM/Fidia and Phoenix network. All devices are labelled with a VAN naming convention. The “VAN_AD” is used to fulfil an automation task while the “VAN_AP” is used to realise a connection between both VAN-ADs. The objects “Path Element” on the left and middle of Figure 16 are required for each device. These objects are used to store information of

specified QoS requirements for each device and define the physical routing. Additionally each “Path Element” connects the device with the corresponding network. The “Network Segment” objects on the left and right side of the figure represent the network segments where devices can directly communicate with each other. Nevertheless, the VAN-APs break up this constellation. The objects “Profinet” and “DSL” in the middle and on the left of the figure represent characteristics of the according network technology layer class of the VAN Information Model. These objects describe technical requirements of the used communication technology.

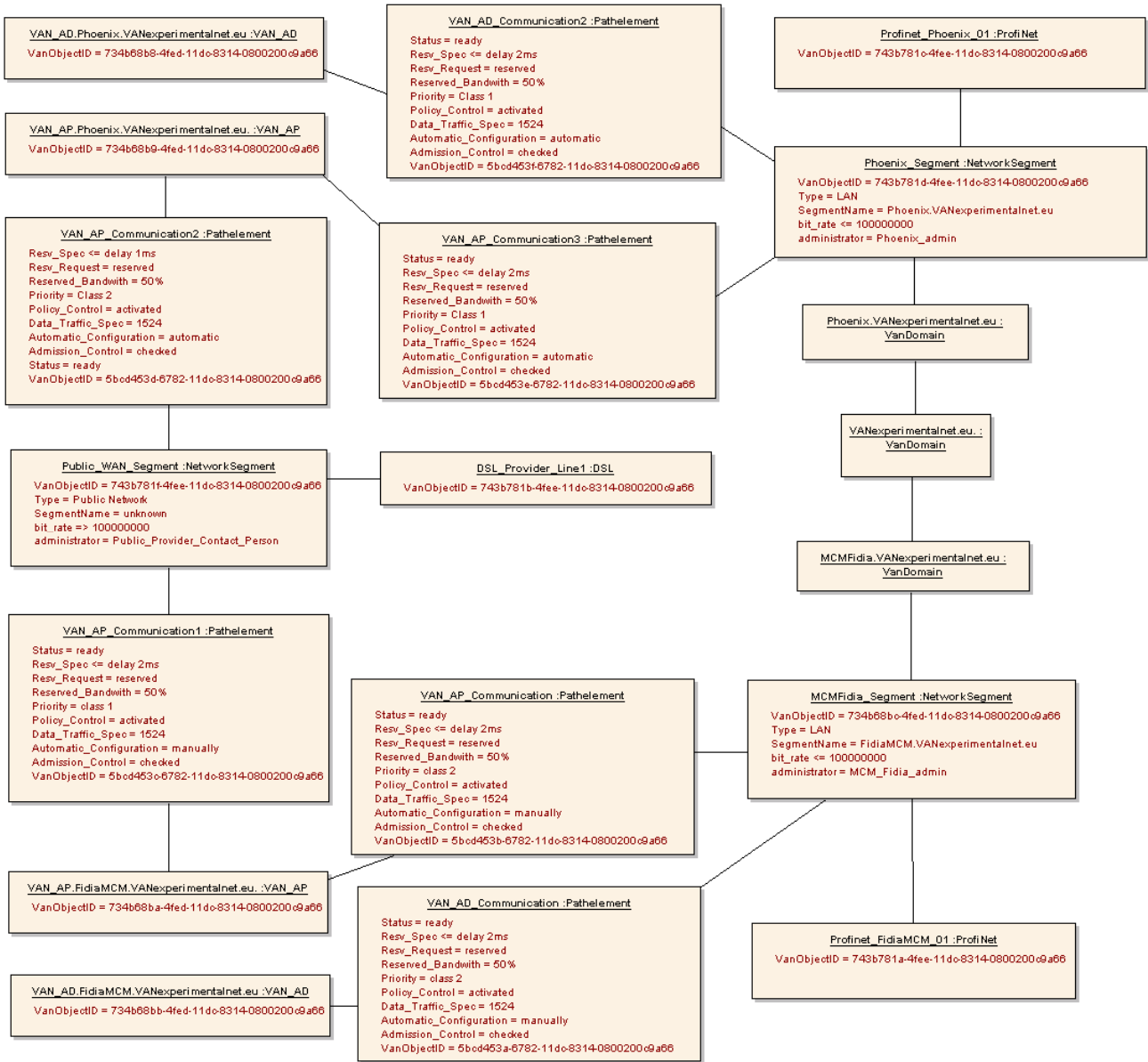


Figure 16: UML Representation of VAN Instance Model

4 VAN Device Description

4.1 VAN Device Description Schema

The scope of this chapter is to specify a generic schema for the VAN device description (VAN-DD). It contains basic elements, which are needed for the generation of a VAN-DD of a specific device, and contains also mechanisms for the extension of new ASEs or ASE classes, which will be specified by other work packages and which are not available at the moment. The VAN-DD schema can also be downloaded from the VAN Groupware [VAN Group].

Vendors can use this schema specification for the description of their own VAN devices. Therefore, they generate a VAN device description file for each device. This file provides common information as well as supported ASEs, ASE classes and attributes with their default values. This VAN-DD is used for including a concrete device instance into the VAN Engineering Tool and for the parameterisation of the device's attributes. The parameterisation values are stored in a repository outside the device description. This may be a VAN-DD instance file –as it is used for the engineering prototypes within this project– or a database. For the engineering prototypes, which will be specified in this work package, one VAN-DD instance file for each device is generated and handled by the engineering tool, but other repositories like databases are also applicable. Detailed information about the use of a VAN-DD and the generation of instance information is provided in chapter 3 of deliverable D08.4-1.

It has also to be mentioned that the VAN-DD only contains the VAN specific configuration elements for engineering, e.g. common device information and the specification of ASEs and ASE classes. Furthermore, another VAN-DD file for the used communication technology must be provided. E.g. for every Profinet device, a GSD file must be available. The cause of this splitting was explained in the deliverable D08.3-1.

4.1.1 Used XML Schema Elements

The VAN-DD schema is based on XML. The schema file is labelled with the file extension ".xsd". The following constructs are used for the specification:

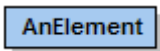
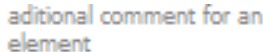
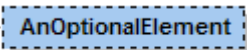

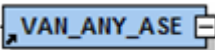
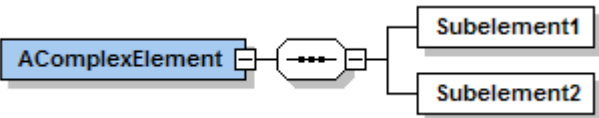
Name	Picture	Description
Element		An element is an entity, which can be described within a schema. Regarding VAN, an element can be e.g. an ASE, a class, etc. Additionally it can have data types, attributes, cardinalities, etc.
Comment		For a more detailed explanation of an element, comments can be used.
Optional Elements		Elements can be optional, i.e. they can be instantiated or not. Optional elements are represented by dashed lines.
Cardinality		Elements can have cardinality. That means, it can be defined how often it can be instantiated. The cardinality is specified on the lower right corner below the symbol.
Reference		Elements can be referenced in other complex schema elements. The reference is represented by an arrow on the lower left side of the symbol.
Complex Elements		Elements can have a complex structure. Therefore, it can be represented in a hierarchical manner. This is realised by a "sequence", "choice" or "all" connector. E.g. the according picture shows a sequence connector.
Schema Import		A schema may be assembled by several schemas. There, the import mechanism can be used. The imported schemas must not belong to the same namespace.

Table 3: Used Elements for the Specification of the XML Schemes

Table 3 shows the mostly used elements for describing the VAN-DD schema. Of course, all other entities, which are specified for XML schemes by the W3C, can be used, e.g. restrictions, enumerations, etc. [W3ORG].

4.1.2 Rules and Naming Conventions for Schema Elements

For getting a homogeneous presentation of all VAN-DD schema's elements, the following rules should be met:

1. Naming conventions:
 - a. ASEs and class names should be specified with capital letters. For the connection of multiple words underlines should be used
 - b. Attributes should be specified with small letters. Exceptions for common terms like "ID" or "IP" are allowed. For the connection of multiple words hyphens should be used.
2. Attributes can have data types according to deliverable D02.2-2. Default or fixed values can be predefined.
3. Attributes can have a complex structure. This allows the implementation of hierarchical structures, e.g. as specified for device-description-specification in Table 3.

4.1.3 Update and Extension

As mentioned above, new ASEs or ASE classes will be specified until completion of the VAN project. Moreover, it must be ensured that new devices can be described within this schema after project termination. Within VAN, there will be a continuously updated specification with the valid information on the VAN Groupware, [VAN Group].

To extend the VAN-DD schema a modular assembly of the VAN-DD is chosen. Thus, an own schema file for each ASE class needs to be specified. The name of the schema file should have the following structure: CLASS_ClassName.xsd. An example for this is presented for the Device Config ASE in chapter 4.1.5.

As shown in Figure 17 two activities must be executed. Firstly the ASE class must be implemented as an XML schema according to the specification in VAN deliverable D02.2-2. The second step is the inclusion into the hierarchically higher schema file. Therefore, the schema must be imported and referenced in the according schema file.

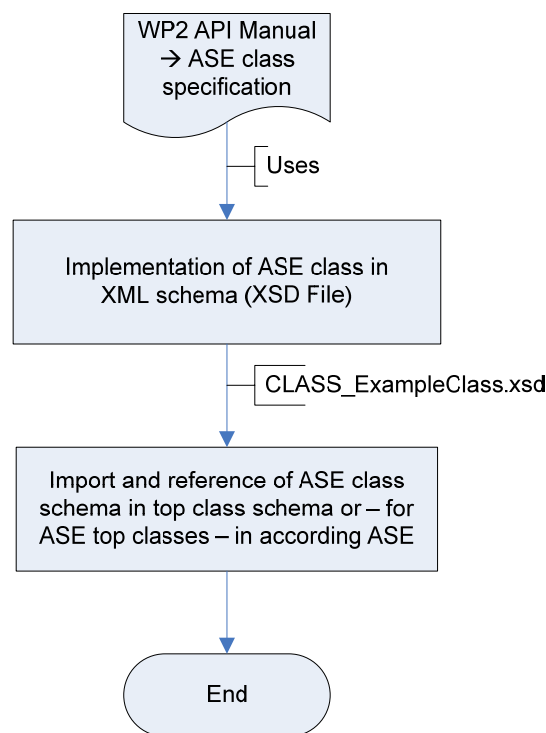


Figure 17: Specification of new ASE Class Schemes

4.1.4 Generic Structure of the VAN-DD Schema

In this subchapter, the main structure of the schema specification is presented. The schema contains two main elements, the DD-Header and the Supported-ASEs, as shown in Figure 18.

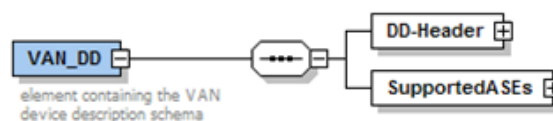


Figure 18: VAN-DD Main Elements

The DD-Header element provides common information, which is needed for the administration within the engineering environment. It does not contain any data of an ASE. The content of the header is shown in Figure 19

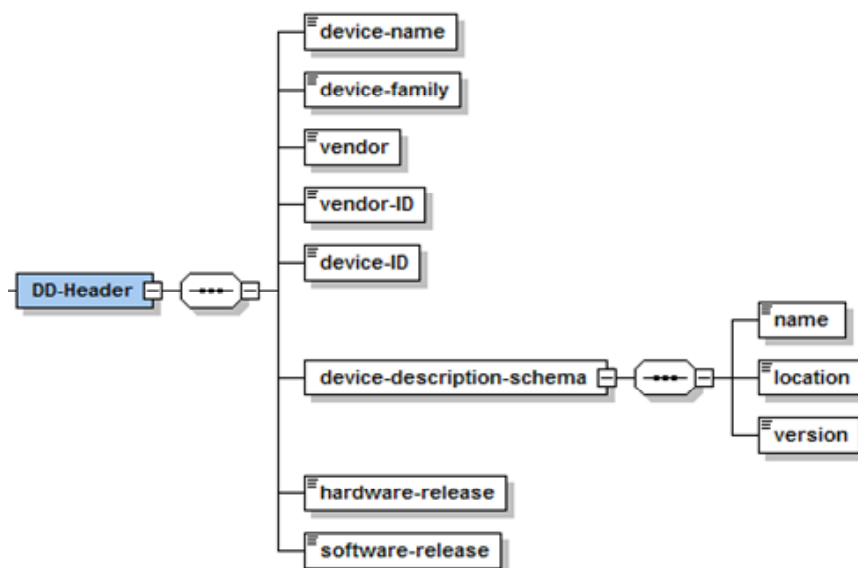


Figure 19: VAN-DD Header Element

All attributes are mandatory. They provide the following information:

Name	Type	Description
device-name	string	Contains the name of the device, which is described by this VAN-DD file
device-family	string	The VAN device family is according to the defined types in [D02.2-1], e.g. VAN-AP or VAN-AD.
vendor	string	Contains the name of the vendor.
vendor-ID	string	Contains the vendor-ID. The ID must be unique and should be franchised by the VAN consortium.
device-ID	string	The device-ID is a unique ID for all devices of a vendor. The vendor has to keep this ID unique.
device-description-schema	string	This is a complex element and provides the name, location and version about the schema, which is used for generating the VAN device description file.
name	string	This element contains the name of the schema.
location	URI	This element contains the URI of the used schema.
version	string	This element gives information about the version of the used schema.
hardware-release	string	The hardware release specifies the version of the delivered hardware.
software-release	string	The software release specifies the version of the delivered firmware.

The `SupportedASEs` element contains the generic ASE element as specified in chapter 4.1.3. It is shown in **Fehler! Verweisquelle konnte nicht gefunden werden.** When this element is used for a VAN-DD, all ASEs, which are implemented in the VAN device, are included here. The source of the ASE specification is the API Manual [WP2 API], which is available on the VAN groupware [VAN Group]. As specified in the deliverable D02.3-1. For a minimum configuration the *VAN Device Config* ASE and the *VAN Domain* ASE must be implemented in every VAN device.

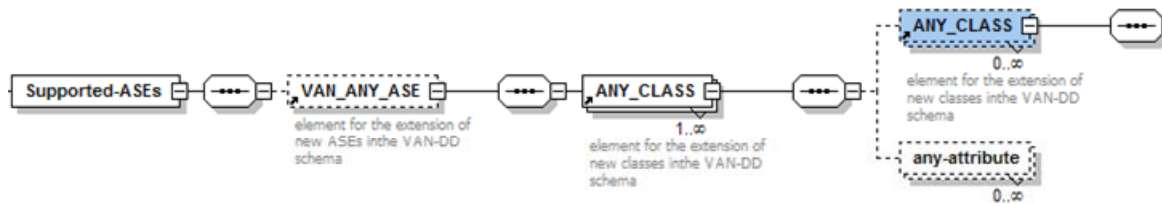


Figure 20: Supported ASEs Element

Every ASE contains at least one ASE class. A class beyond can contain attributes and other classes. This mechanism permits the assembling of hierarchical structures according to the deliverable D02.2-2. The specification of each ASE is realised in a separated XML schema. Each ASE's schema is stored on the VAN Groupware [VAN Group]. Thus, a flexible extension of the specification by the responsible work packages is achieved.

4.1.5 Implementation of new ASEs or ASE Classes

Because of the continuous extension by defining new ASEs or ASE classes in this project, a flexible integration must be possible. Thus, an ASE is composed by several schemas by using the import mechanism. The following rules have to be met:

1. For every ASE an own schema must be defined.
2. For every ASE class an own schema must be defined.
3. Every ASE must have at least one class.
4. Every ASE class inherits the attributes "member-in-groups", "object-reference" and "object-type" from the root element VAN_ASE. According to deliverable D02.2-2 for every class the attributes "VAN_ASE", "CLASS", "CLASS_ID" and "PARENT_CLASS" have to be specified. These attributes must have fixed values.
5. An ASE class must be integrated into the parent class or, if it is a top class, into the ASE schema by using the import mechanism. Moreover, it must be referenced in the imported schema.

Following the composition of the *VAN Device Config* ASE is described. This ASE contains two classes, `CLASS_COMMON_DEVICE_CONFIG` as top class and the `CLASS_DEVICE_DESCRIPTION`. The ASE schema file consists of three main elements see also Figure 21

import	oc:CLASS_COMMON_DEVICE_CONFIG.xsd
complexType	ASE_DEVICE_CONFIG_BASIC
element	DEVICE_CONFIG_ASE

Figure 21: Main Elements of Device Config ASE

Moreover, the imported ASE class must be referenced within the `DEVICE_CONFIG_ASE` element. This is shown in Figure 21. The ASE class `DEVICE_DESCRIPTION` is imported and referenced in the schema of `CLASS_DEVICE_DESCRIPTION` analogously.

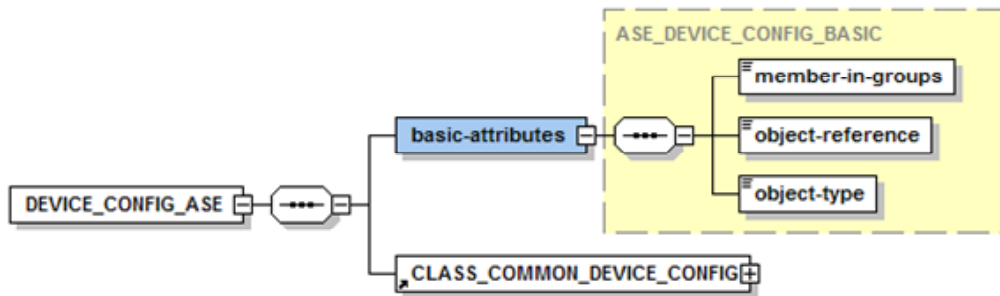


Figure 22: Inclusion of an Imported Element

Figure 23 shows the whole VAN Device Config ASE. The actual specification of this ASE describes two classes. Regarding the hierarchy of the ASE the class `COMMON_DEVICE_CONFIG` is implemented on the root level. This class provides a number of mandatory attributes, e.g. `class-version`, optional attributes like the `factory-reset` and further a complex element `list-of-device-config-objects`. This element includes a list of objects, which are used for the actual configuration of the related VAN device.

Furthermore, a second class `CLASS_DEVICE_DESCRIPTION` is related to `CLASS_COMMON_DEVICE_CONFIG` via the attribute `TOP_CLASS` via the inheritance mechanism defined in deliverable D02.2-2. Thus, it is defined as child element in the VAN-DD schema.

As mentioned in deliverable D02.2-2 there will be additional classes specified within this ASE, e.g. for wireless communication or real time. These schemas must be included analogously.

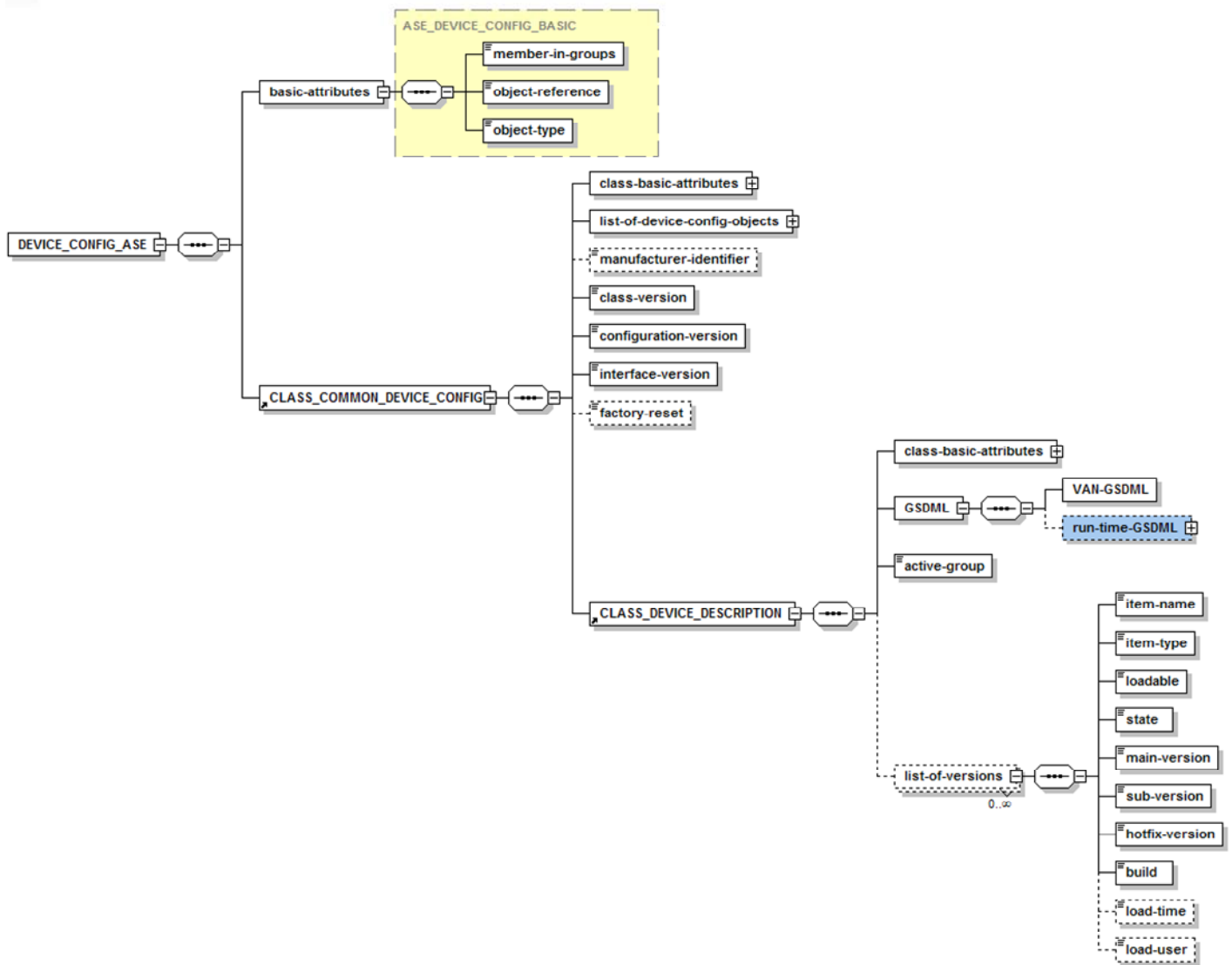


Figure 23: Example- VAN Device Config ASE

4.1.6 Naming Conventions of a VAN Device Description File

Regarding a better administration of VAN-DD within an engineering environment, a formal naming convention for each file should be specified. The unambiguous name should provide information about VAN device type, vendor as well as for the version handling. The name should contain the following data;

VANDD-[vendor name]-[device name]-[VAN device type]-[date].vdd

The words in the brackets have the following meaning:

- The *vendor name* is according to the name of the vendor. Hyphens and blanks are allowed.
- The *device name* is specified by the vendor. Hyphens and blanks are allowed.
- The *VAN device type* is according to the defined types in [D02.2-1], e.g. VAN-AP or VAN-AD.
- The *date* parameter describes the date of release. It has to be presented in the following manner: *yyyymmdd*
- Furthermore, the file extension should be ".vdd".

Following an example for a valid VAN device description file name is provided:

VANDD-EnterpriseXYZ-MyVanDevice-VAN_AD-20070727.vdd

4.2 VAN-DD for VAN Devices

A VAN-DD provides type information of the capabilities of a VAN device. It is comparable with a GSD file for PROFIBUS DP devices. The basis for the generation of a VAN-DD is the specification of the schemas, which are introduced in the previous chapter. The VAN-DD can be interpreted by a VAN-ECD. Within the VAN-ECD the included type information is instantiated for configuration and parameterization of a real device, which must be engineered for running within a VAN domain.

In Figure 24, a structure of a VAN-DD is provided. It specifies, that this device can be configured with three ASE objects with several classes. The VAN device description file is used for generating two instances within a VAN-ECD. For the first, *EXAMPLE_Instance_1*, merely the *Device Config ASE* and the *Domain ASE* with a subset of the available classes are configured, whereas for the second, *Example_Instance_2*, all ASEs with a subset of available classes are used.

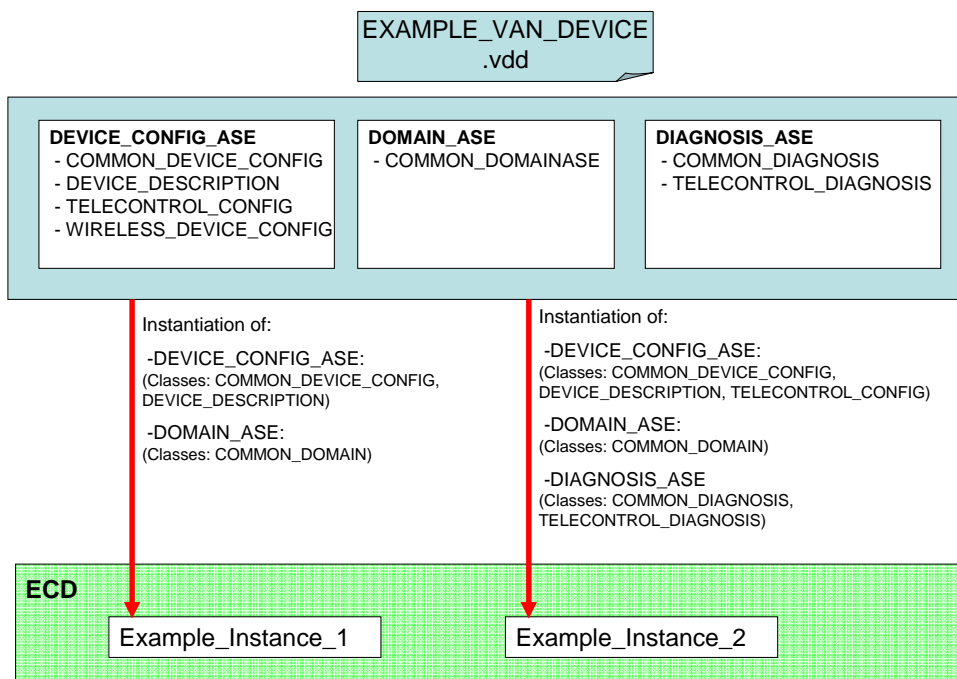


Figure 24: Instantiation of a VAN Device Description File

This example shows that it is possible to instantiate only a subset of the provided ASEs or ASE classes, which are specified in a VAN-DD. But it is always mandatory, when instantiating an ASE class, that every hierarchically higher class must be instantiated too. E.g. if the class *DEVICE_DESCRIPTION*, which is specified within the *VAN_DEVICE_CONFIG_ASE*, needs to be included, the class *COMMON_DEVICE_CONFIG* must be instantiated as well. Figure 25 shows an example of a VAN-DD. The file with this VAN-DD is also available on the VAN Groupware as a reference implementation [VAN Group]. The file contains a schema for the minimal configuration of a VAN device according to conformance class A, which is defined in deliverable D02.3-1. Therefore, the *Device Config ASE* and the *Domain ASE* are included. The whole VAN-DD is assembled analogous to chapter 4.1.5 by using the import and reference mechanisms.

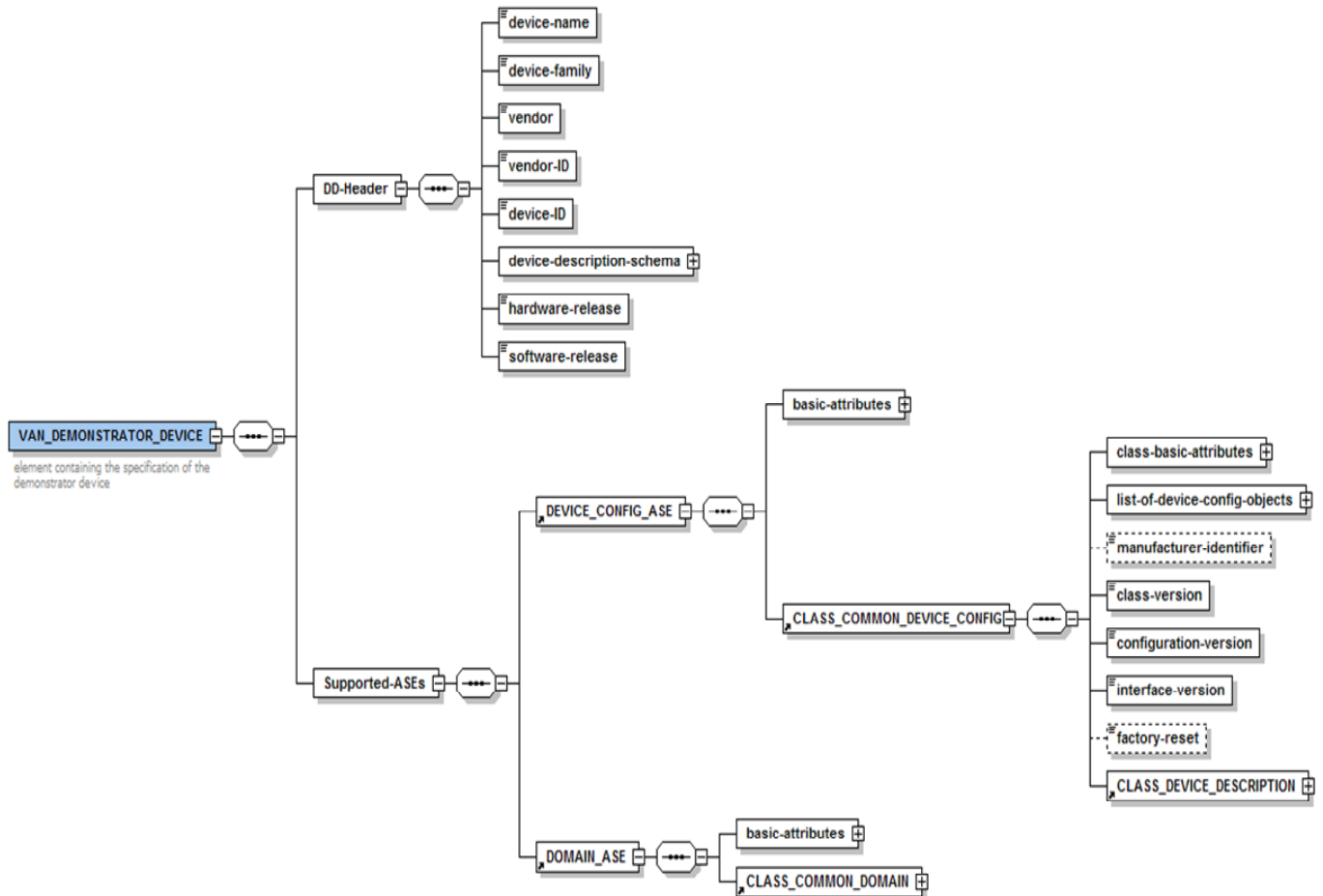


Figure 25: Example Schema of a VAN Device Description File

4.3 VAN-DD Instance File for a VAN Device

Based on a VAN-DD, i.e. based on the schema specification of the capabilities of a VAN device, a VAN-DD Instance File can be generated with the engineering tool. The VAN-DD instance file contains actual configuration and parameterization data. The data are sent to the VAN device via web services. For the VAN engineering demonstrator one VAN-DD instance file per device configuration is stored in a file repository.

An example of a VAN-DD instance file is shown in Figure 26. It is an instantiation of the VAN device description file example, which was introduced in chapter 4.2 and is also available on the VAN Groupware [VAN Group].

The screenshot displays an XML editor interface for a file named 'VAN_DEMONSTRATOR_DEVICE'. The document is edited with XMLSPY v5 rel. 4 U. The root element is 'VAN_DEMONSTRATOR_DEVICE', which contains several sub-elements and attributes:

- Namespaces:**
 - `xmlns:xsi` pointing to `http://services.van.ovgu.de/`
 - `xsi:noNamespaceSchemaLocation` pointing to `http://services.van.ovgu.de/VANDD-VAN Consortium-DemonstratorDeviceExample-VAN_AD-20070921.xsd`
- DD-Header:**
 - `device-name`: VAN Demonstrator Device
 - `device-family`: VAN_AD
 - `vendor`: The VAN Consortium
 - `vendor-ID`: 0x07
 - `device-ID`: 0x01
 - device-description-schema:**
 - `name`: VAN DD Schema
 - `location`: `http://services.van.ovgu.de/`
 - `version`: V1.0
 - `hardware-release`: V1.1
 - `software-release`: V1.1
- Supported-ASEs:**
 - DEVICE_CONFIG_ASE:**
 - basic-attributes:**
 - CLASS_COMMON_DEVICE_CONFIG:**
 - class-basic-attributes:**
 - `list-of-device-config-objects`:
 - device-config-object (1):**
 - `class-version`: 1.0
 - `configuration-version`: 1
 - `interface-version`: 1.0
 - CLASS_DEVICE_DESCRIPTION:**
 - class-basic-attributes:**
 - `GSDML`:
 - `active-group`: 5
 - DOMAIN_ASE:**
 - basic-attributes:**
 - `member-in-groups`: 5
 - `object-reference`
 - `object-type`
 - CLASS_COMMON_DOMAIN:**
 - class-basic-attributes:**
 - configuration-data:**
 - `link-id`: ID0123456789ABCD
 - `local-name`: www.van.ovgu.de
 - `remote-name`: www.van.ovgu.de
 - `master`: active
 - `local-ip`: 127.0.0.1
 - `remote-ip`: 255.0.0.1
 - `local-mac`: 00-00-00-00-00-00
 - `remote-mac`: 00-00-00-00-00-00
 - operation:**
 - `status`: Connected
 - `command`: Setup

Figure 26: VAN-DD Instance File

4.4 GSDML for VAN Device

The General Station Description Markup Language (GSDML) is used to define a VAN device in Profinet. This is necessary, because only with a GSD the VAN device can be visualized and engineered in the local engineering tool (Step7). In the GSD all Profinet-IO aspects and properties of the device are described. GSD is improper for the description of technological functions or graphic user interfaces of the device. GSD describes exactly the device model of Profinet-IO, consisting of places of insertion (slots) and groups of I/O channels (sub slots). Slots and sub slots represent the addressing information of data inside a module.

4.4.1 Introduction to GSD

The GSD is used in the local engineering tool (Step7) to build and maintain the hardware catalogue of available devices and to define the variables, function and communication features of the devices. As defined in deliverable D08.3-1 the VAN device description references the GSD, as shown in Figure 27.

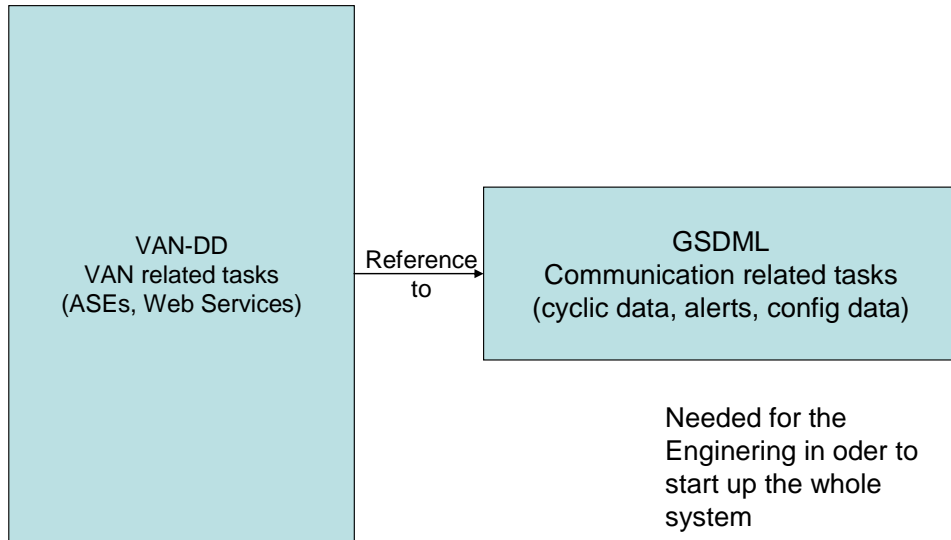


Figure 27: VAN-DD Schema References GSD

4.4.2 Structure of a GSD

Figure 28 shows the top level structure of a GSD . The root element shall be "ISO15475Profile"

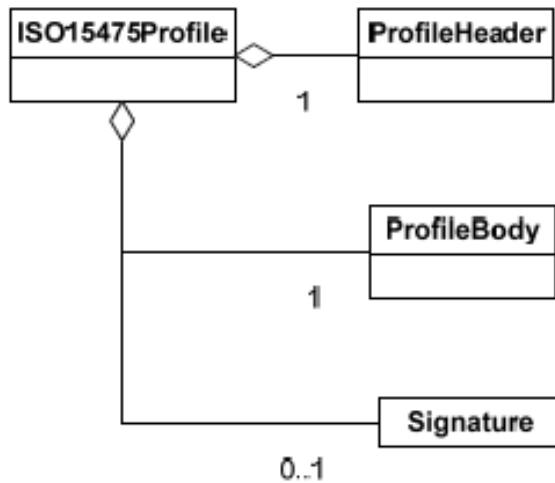


Figure 28: The Top Level Structure

4.4.3 Useful parts for VAN Devices

Figure 28 shows the top-level structure of a GSD. The root element shall be "ISO15475Profile". The VAN device description should reference a GSD file. In the schema of the *ProfileBody* the element *ProductFamily* is not used, because the attribute makes no sense for each module. Thus, the attribute is in the prototype used as a link to a VAN device.

5 VAN Tool Calling Interface

For integration of Profinet IO devices in an engineering system, GSD files are used which must be provided by the device manufacturer. Syntax and semantics of these files are standardised, so that the devices can be integrated independently of the manufacturer.

It is required for some equipment to provide a separate tool for parameter setting and diagnostics if the requirements for data input or visualization go beyond the scope of GSDML means.

The user of the Engineering System is faced with the following problems:

- He must have knowledge of which tool is required for a particular device
- He has to enter the communication parameter of the device both in the Engineering System and in the device specific tool (Device Tool) and make sure to keep the parameter consistent.
- He must take care that both, Engineering System and Device Tool, use the same location for data storage because otherwise it will be difficult to archive the project data.

From the device manufacturer point of view, the following problems must be solved:

- He has to implement the communication functionality for each supported field bus system.
- A communication relation cannot be established by the tool if the target device is located in a different network and only a proprietary gateway interconnects the networks.

In order to fix these problems, a relation between the GSD based Engineering System and the Device Tool is needed. This relation should cause only minimal dependencies between the involved tools. Moreover, an easy version handling is required to ensure the interaction between different versions of the interface.

This interface is defined as TCI (Tool Calling Interface) chapter 2.1.2 which covers Profinet.

5.1 Program Interface Description for VAN

The Program Interface Description (PID) file describes the properties of the Device Tool and contains data which are needed by the Engineering System to build menu entries in its GUI. The PID file is a XML file. This file shall be provided by the manufacturer of the Device Tool and should be installed by the installation program of the Device Tool. This program shall also insert the name and installation path in the system registry.

The PID file allows the Engineering System to extend its GUI menu structure with the name of the Device Tool so that the user is able to start the Device Tool for example from the context menu of a selected device.

Figure 29 shows how a context menu of the Engineering System can be extended by the name of the Device Tool.

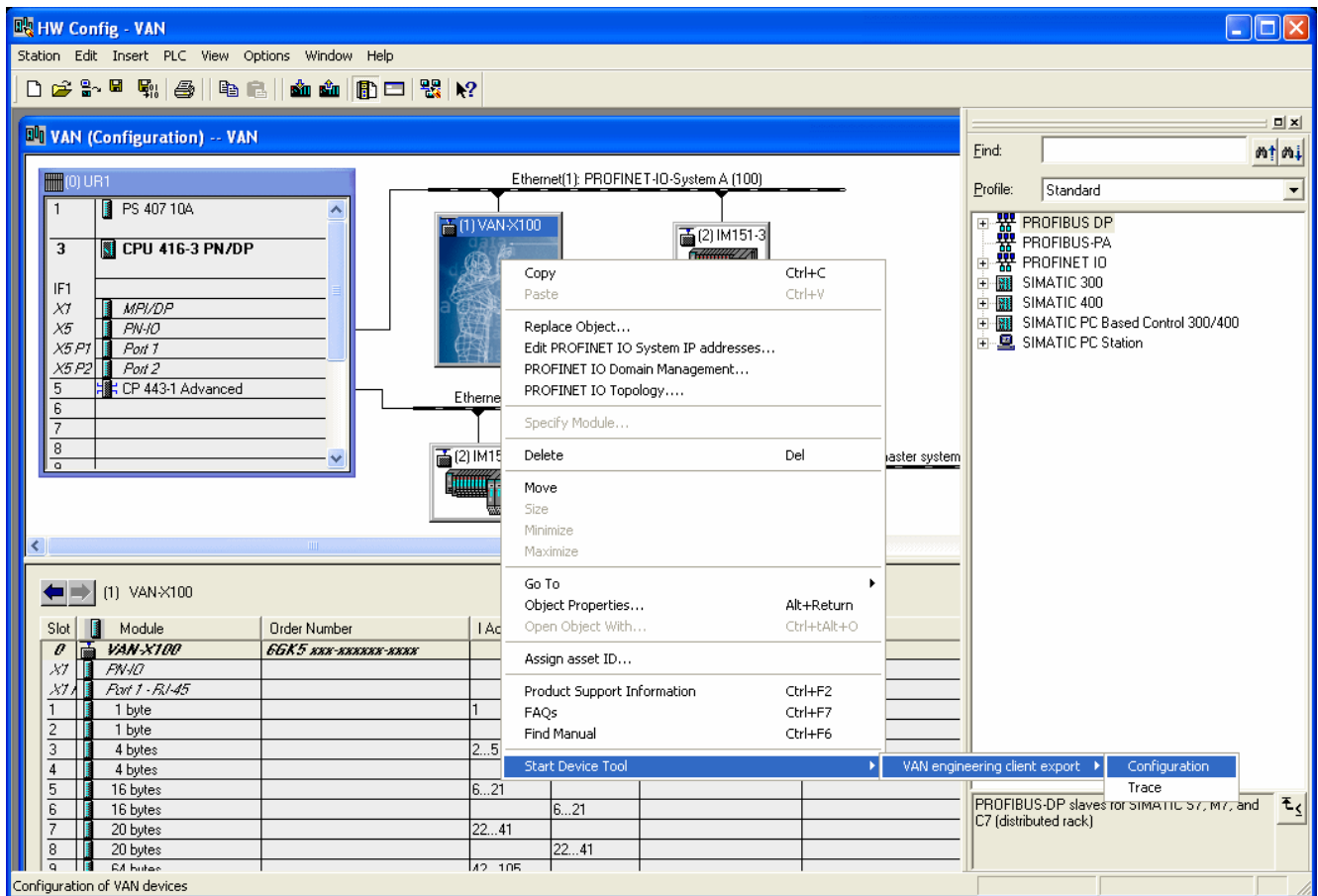


Figure 29: Context Menu through PID File

5.1.1 Example of a PID File

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

```
<ProgramInterface
```

```
  xmlns="http://www.profibus.com/TCI/2006/03/PID"
```

```
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
  xmlns:prim="http://www.profibus.com/TCI/2006/03/Primitives"
```

```
  xsi:schemaLocation="http://www.profibus.com/TCI/2006/03/PID ..\XSD\TCI-PID-V0.9.xsd">
```

```
  <prim:DocumentInfo Version="1.0"/>
```

```
  <General VendorName="Siemens AG - Example">
```

```
    <ToolDescription ToolName="VAN engineering client export"
      ToolDescription="Export of device description"
      xml:lang="en"/>
```

```
    <ToolDescription ToolName="VAN Engineering Client Export"
      ToolDescription="Exportieren der Device Description"
      xml:lang="de"/>
```

```
    <InvocationPrefix name="c:/tmp/van.xml"/>
```

```
  </General>
```

```

<ConformanceClass Name="C2">
  <OptionalFeature Name="UsesMultipleDeviceInformation"/>
  <OptionalFeature Name="SupportsObjectDeletion"/>
</ConformanceClass>

<EntryPoints>
  <EntryPoint ID="1">
    <InfoText EntryPointName="Configuration"
      EntryPointDescription="Configuration of VAN Devices"
      xml:lang="en"/>
    <InfoText EntryPointName="Konfiguration"
      EntryPointDescription="Konfiguration von VAN Devices"
      xml:lang="de"/>
  </EntryPoint>
  <EntryPoint ID="2">
    <InfoText EntryPointName="Trace"
      EntryPointDescription="Show the trace buffer in detail"
      xml:lang="en"/>
    <InfoText EntryPointName="Trace"
      EntryPointDescription="Anzeige der Traceausgaben"
      xml:lang="de"/>
  </EntryPoint>
</EntryPoints>
</ProgramInterface>

```

The PID file allows the local engineering tool, for example Step7, to extend its GUI menu structure with the name of the Device Tool. Then the user is able to start the Device Tool for example from the context menu of a selected device. In the example shown in Figure 29, two menu entries *Configuration* and *Trace* are generated, which are enabled in the local engineering tool if the user select the VAN device context menu entry.

5.2 Architecture of TCI

The Engineering System and the Device Tools are installed on the same PC, running Microsoft Windows operating system. Each tool is installed separately.

Within the Engineering System, a Profinet-IO-Device is configured with the help of the corresponding GSDML file of the device manufacturer. This step includes assignment of communication addresses (IP-Address/NameOfStation for Profinet IO), placing of modules in case of a modular device and adjustment of the device parameter defined in the GSDML.

The TCI standard allows associating Device Tool identification with Profinet device identification. The Engineering System uses TCI specified mechanism to find the Device Tool for a given device. The Engineering System provides, e.g. in the context menu of a selected device or a module, an entry that can be used to start the Device Tool.

After the Device Tool is started, it identifies the selected device and the user can instantly establish a communication with the device without having to enter address information in the Device Tool.

5.3 VAN Temporarily Parameter File

The Temporary Parameter File (TPF file) is used from the TCI to pass parameter for the Device Tool. Additional parameters can be defined in the local engineering tool through attribute value pairs. The predefined values in the GSD and the additional defined values in the local engineering tool will be written to a TPF file. The Device Tool is started to use this TPF file. The schema of the TPF file is

defined in the TCI specification of Profinet [TCI]. The Device Tool shall delete the TPF file after its operations are concluded.

5.4 General Structure Definition

The structure of the TPF file is shown by an UML-class-diagram in Figure 30 [D08.4-1]. This schema is built in a generic way, which means that a new parameter does not require the schema itself to be updated. This allows the introduction of new parameters without a definition of a new TPF file structure.

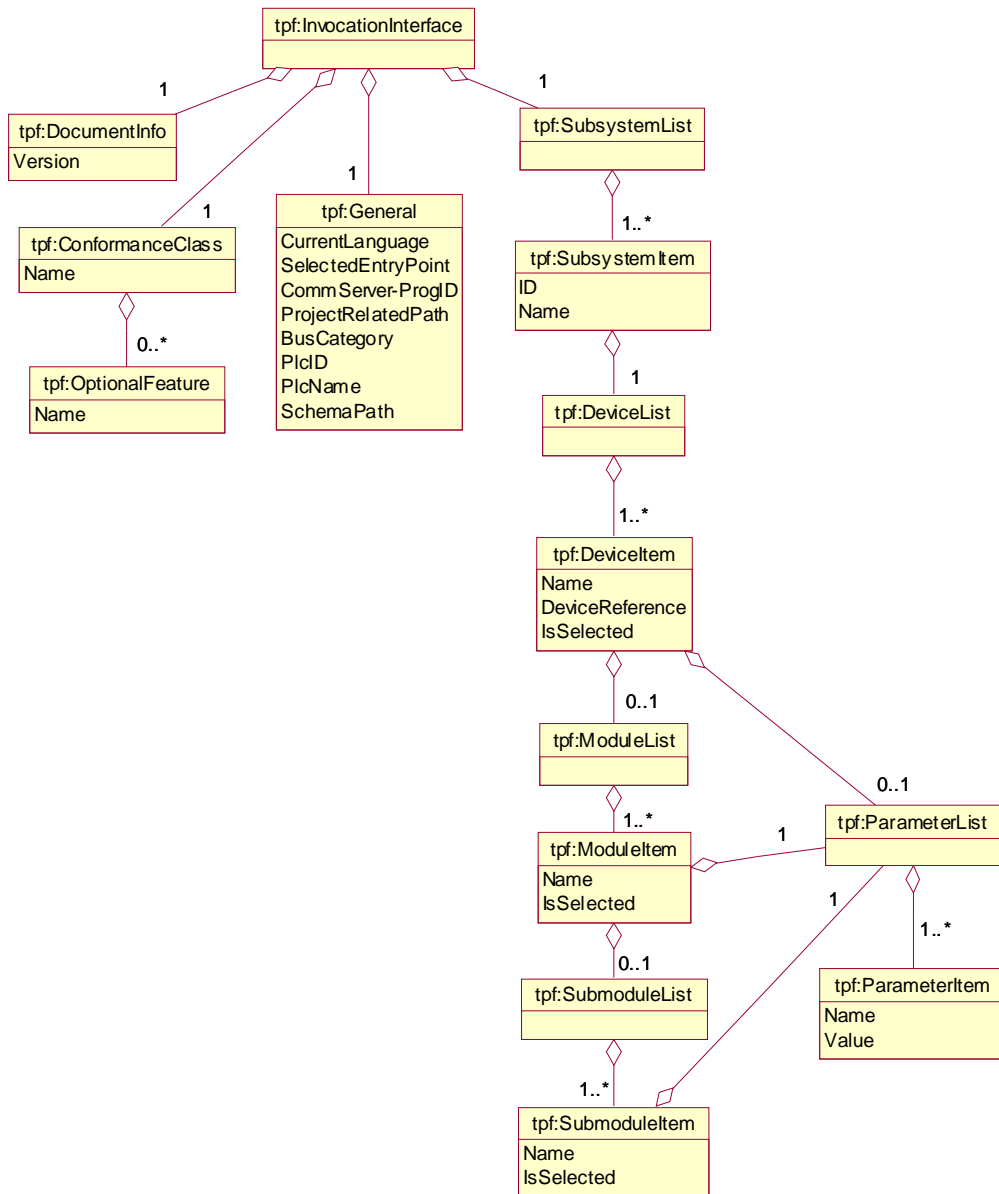


Figure 30: TPF File Structure

6 VAN Field Device Tool

In this chapter, a further description of the interfaces used to realize the Field Device Tool (FDT) of the integrated concept for a VAN Engineering Tool will be provided. The chapter is mainly divided in two main sections, one dedicated to the VAN Device DTM and one related to the VAN Communication DTM. These two entities must be close coupled to provide a reliable solution for the configuration of VAN devices.

6.1 VAN Device DTM

The VAN Device DTM shall be used for the configuration and maintenance of VAN devices. From the VAN point of view, only the provided ASEs shall be modified. Therefore, the VAN Device DTM has to be adapted to the real VAN device by means of a VAN device description and with provided device instance information, passed via TPF file in context of TCI call or by means of a dedicated VAN-DD instance file. The device instance data will be shared in VAN context, e.g. to be used in several engineering tools.

6.1.1 Supported Interfaces

This sub chapter defines the provided interfaces of the VAN Device DTM and gives an overview about the offered possibilities of the DTM regarding the interacting user. Furthermore the allowed actions in relationship to the operation phase of the DTM or the whole system will be defined.

The VAN Device DTM has to provide the required FDT-Interfaces, defined in the FDT specification [FDT1.2.1]:

- IDtm
- IDtm2
- IDtmImportExport

To build an export image of a DTM a Frame Application uses one IStream object for each device instance. This IStream object is used as argument to IDtmImportExport::Load() or IDtmImportExport::Save().

In contrast to the normal usage, the VAN Device DTM has to store the relevant information inside the VAN-DD instance file. Extra data has to be stored and imported via this interface.

- IDtmActiveXInformation
- IDtmInformation
- IDtmInformation2
- IDtmOnlineParameter
- IDtmParameter
- IFdtCommunicationEvents
- IFdtCommunicationEvents2
- IFdtEvents

This interface allows a Frame Application the online access to a device. This interface is mandatory for all devices which must be loaded during commissioning.

AuditTrail according to the FDT specification will not supported.

6.1.2 User Roles

It is expected that a DTM adapts the provided list of functions according to the role of the current user [FDT1.2.1]. As long as the DTM does not have valid user information (e.g. in state "running"), it should behave like the user with the least rights ("Observer") is using the DTM.

The following tables will give a brief description of the actors' roles and the supported ASEs. The listed ASEs represent the recent state of the project, especially the defined ASE types. Therefore, the tables should be enhanced, if several ASEs are finally defined.

Actor Name	Observer
Brief Description	This actor stands for a person that may only observe the current process.
Access Verification	The access as "Observer" actor may not have a password.
Supported ASEs	<ul style="list-style-type: none"> • VAN_Device_Config <ul style="list-style-type: none"> • Get_DeviceConfig() • VAN_Domain <ul style="list-style-type: none"> • Get_Domain()

Table 4: Role 'Observer'

Actor Name	Operator
Brief Description	This actor stands for a person who has to observe and manage the current process. The "Operator" may therefore check the current status of the device, modify set values and check if the device is well functioning. The use cases for this actor role should enable the user to perform a complete diagnosis, watch the actual status and parameter set as well as the current process variables.
Access Verification	The access as "Operator" requires a password.
Supported ASEs	<ul style="list-style-type: none"> • VAN_Device_Config <ul style="list-style-type: none"> • Get_DeviceConfig() • VAN_Domain <ul style="list-style-type: none"> • Get_Domain()

Table 5: Role 'Operator'

Actor Name	Maintenance
Brief Description	<p>A "Maintenance" person should have the possibility to perform all necessary maintenance operations including device exchange, teaching, calibration, adjustment.</p> <p>The person may therefore download verified parameter sets, modify a subset of parameters online or offline, perform device-specific online operations and at the end of the processing, may have the possibility to upload the complete parameter set.</p>
Access Verification	The access as "Maintenance" requires a password.
Supported ASEs	<ul style="list-style-type: none"> • VAN_Device_Config <ul style="list-style-type: none"> • Get_DeviceConfig() • Set_DeviceConfig() • VAN_Domain <ul style="list-style-type: none"> • Get_Domain() • Set_Domain()

Table 6: Role 'Maintenance'

Actor Name	Engineer
Brief Description	<p>The actor "Planning Engineer" stands for person like a plant engineer, a specialist [VDE 2187] or any fully authorized person. This Person has access to the complete set of functions of the Frame Application and can use DTM functions without any restrictions. Only OEM-specific operations are locked.</p>
Access Verification	The access as "Engineer" requires a password.
Supported ASEs	<ul style="list-style-type: none"> • VAN_Device_Config <ul style="list-style-type: none"> • Get_DeviceConfig() • Set_DeviceConfig() • VAN_Domain <ul style="list-style-type: none"> • Get_Domain() • Set_Domain()

Table 7: Role 'Engineer'

Actor Name	Administrator
Brief Description	The "Administrator" actor stands for person that has to perform administrative operations within an engineering environment. He is responsible for adding and removing of software components.
Access Verification	The access as "Engineer" requires a password.
Supported ASEs	<ul style="list-style-type: none"> • VAN_Device_Config <ul style="list-style-type: none"> • Get_DeviceConfig() • Set_DeviceConfig() • VAN_Domain <ul style="list-style-type: none"> • Get_Domain() • Set_Domain()

Table 8: Role 'Administrator'

Actor Name	OEM Service
Brief Description	The actor "OEM Service" may perform the complete set of DTM specific functions. OEM specific operation may be accessible to an "OEM Service" actor, like resetting of internal counters and exchanging firmware. The "OEM Service" has only inherited use cases, but the inherited use cases (especially the "Online Operation" use case) may have significant extensions.
Access Verification	The access as "Engineer" requires a password.
Supported ASEs	<ul style="list-style-type: none"> • VAN_Device_Config <ul style="list-style-type: none"> • Get_DeviceConfig() • Set_DeviceConfig() • VAN_Domain <ul style="list-style-type: none"> • Get_Domain() • Set_Domain()

Table 9: Role 'OEM Service'

6.1.3 Relationship between User Roles and ACL

The VAN Access Control Layer (ACL) defined in deliverable D06.1-1 offers the possibility to define the allowed rights for each object inside the VAN device in a fine-grained way. Therewith it is possible to restrict each offered service of an ASE for a specific certificate. The biggest difference between User Roles defined by FDT and ACL of VAN is, that the User Roles approach follows a centralistic access control, managed by the engineering system. Figure 31 [Wiki] shows the ACL approach, which allows also a distributed engineering. Therefore, all access rights are owned by the device itself.

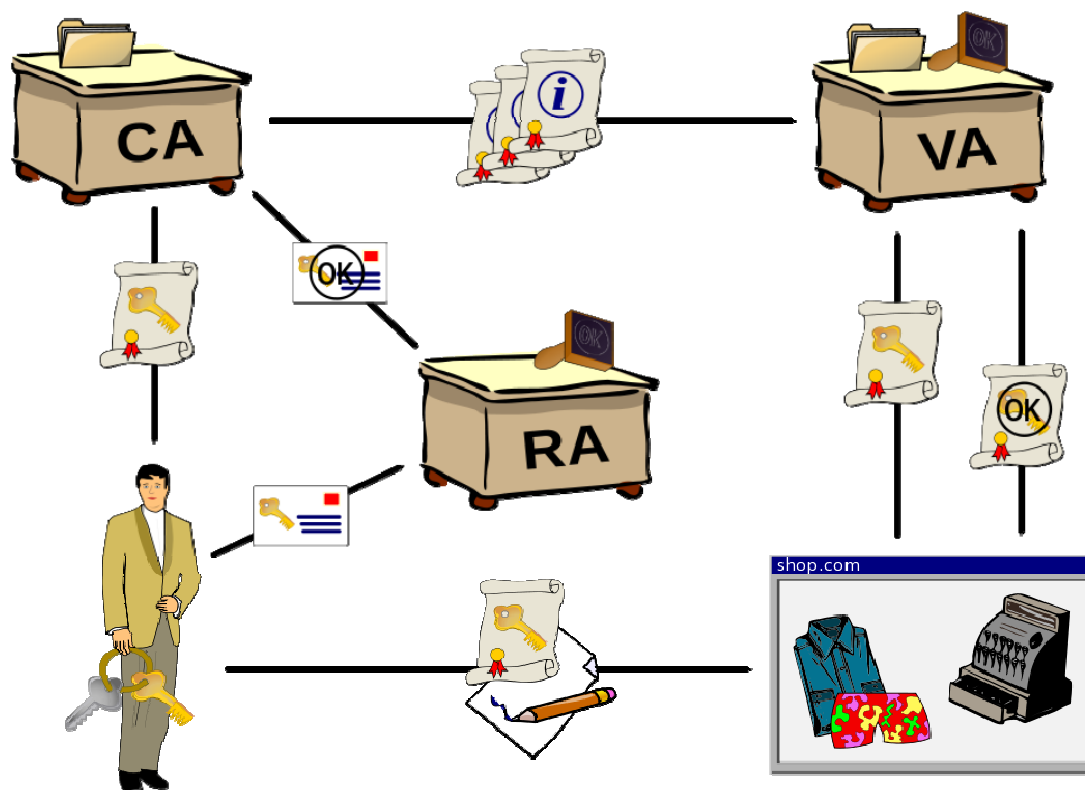


Figure 31: Schema of a Public Key Infrastructure

In order to manage the diversity of access rights, a central Certification Authority (CA) provides specific certificates. These certificates are delivered to both, the user and a central Validation Authority (VA). The Registration Authority (RA) could be optional. It is responsible for the assurance, that the data is integer (e.g. integrity of the public key). If the user / engineering tool would like to communicate with the VAN enabled device, it has to provide following information to the VAN device:

- *WebService to call* Get/Set + ASE name
- *Object Reference* the reference the object used (ASE)
- *Source* the Full Qualified Domain Name (FQDN) of the originator of the request
- *Target* the FQDN of the host addressed
- *Certificate* a reference to the container of the certificate found in the SOAP header

The device can either decide by its own implemented or configured ACLs, if the request for a specific service of an object is accepted or not. In order to reduce the configuration overhead for each device, the device could ask the central VA in order to decide about the acceptance of a certificate.

In a first step, the available engineering tools on the market will not provide such an ACL based access control. Therefore, it would be necessary to map the User Roles to PKI (Public Key Infrastructure) information. One possible solution could be:

Each User Role has assigned a specific certificate. Depending on the user, logged in on the engineering tool, the engineering tool uses automatically the associated certificate in order to communicate with all devices of the automation application.

6.1.4 Representation of ASE

The VAN Device DTM shall use ActiveX controls for the interaction with the user. The VAN Device DTM provides only one default view with all ASEs. Figure 32 shows the standard user interface according to the FDT guideline [FDT_G]. The left area shall be used in order to list all available ASEs. The main area shall be used for the representation of the ASE attribute values. These values shall be editable, if the user has logged on in the appropriate role. Furthermore, the operation phase of the complete engineering tool must be in the correct context.

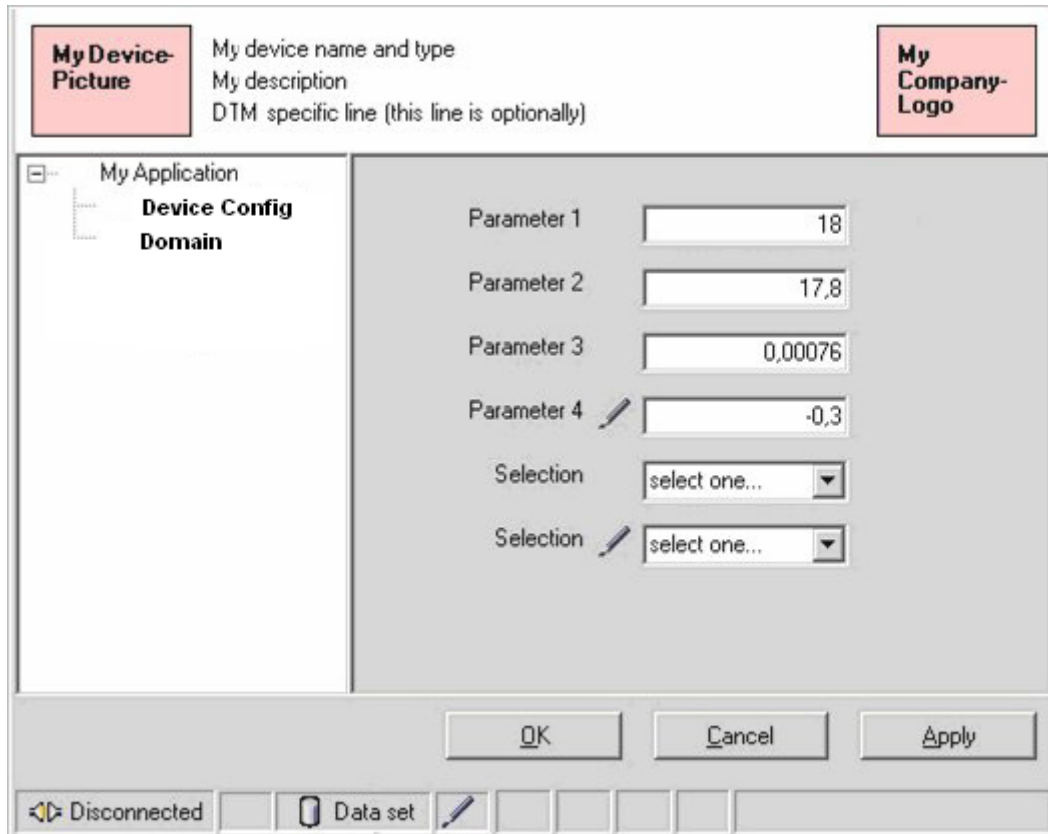


Figure 32: User Interface of a VAN Device DTM

The most efficient way is the representation of the attributes with their names and values in form of a grid. This is also easy to use. A special use case is the representation of attributes of lists. Therefore, the grid shall be able to represent sub grids.

The VAN Device DTM has to fulfil the requirements of the FDT guideline [FDT_G] for colouring, resizeability etc.

6.1.5 Operation Phases

In FDT specification 1.2.1 there are defined the following operation phases from the point of view of the FDT based system [FDT1.2.1].

Phase	Engineering
Brief Description	<ul style="list-style-type: none"> • Planning and configuring of a plant • no online access to the plant
Supported ASEs	<ul style="list-style-type: none"> • VAN_Device_Config <ul style="list-style-type: none"> • Manipulation of all attributes is allowed • VAN_Domain <ul style="list-style-type: none"> • Manipulation of all attributes is allowed

Table 10: Operation Phase 'Engineering'

Phase	Commissioning
Brief Description	<ul style="list-style-type: none"> • Installing the plant and the devices • Scanning the network to verify a planned network • Downloading project data into the plant • Programming, diagnosis of the devices • Adjusting parameters
Supported ASEs	<ul style="list-style-type: none"> • VAN_Device_Config <ul style="list-style-type: none"> • Manipulation of all attributes is allowed • VAN_Domain <ul style="list-style-type: none"> • Manipulation of all attributes is allowed

Table 11: Operation Phase 'Commissioning'

Phase	Runtime
Brief Description	<ul style="list-style-type: none"> • The plant is completely commissioned and running. • Very restricted access to configuration and parameterization data. • Scan to verify the network • Replacing defect devices. • Reading process values and diagnosis information.
Supported ASEs	<ul style="list-style-type: none"> • VAN_Device_Config <ul style="list-style-type: none"> • Reading of all attributes is allowed • Writing of data for replaced devices only • VAN_Domain <ul style="list-style-type: none"> • Reading of all attributes is allowed • Writing of data for replaced devices only

Table 12: Operation Phase 'Runtime'

Phase	Service
Brief Description	<ul style="list-style-type: none"> • This operation phase is used for service tool Frame Applications. • Scan to create a topology. • Scan to verify a network. • All service related operations. • Unrestricted access to the device.
Supported ASEs	<ul style="list-style-type: none"> • VAN_Device_Config <ul style="list-style-type: none"> • Manipulation of all attributes is allowed • VAN_Domain <ul style="list-style-type: none"> • Manipulation of all attributes is allowed

Table 13: Operation Phase 'Service'

Phase	Not supported
Brief Description	<ul style="list-style-type: none"> • The application does not support the operation phases defined above. • A DTM must offer all functions if the Frame Application passes "not supported".
Supported ASEs	<ul style="list-style-type: none"> • VAN_Device_Config <ul style="list-style-type: none"> • Manipulation of all attributes is allowed • VAN_Domain <ul style="list-style-type: none"> • Manipulation of all attributes is allowed

Table 14: Operation Phase 'Not supported'

Before the VAN Device DTM can be used, a VAN-DD must be assigned to a DTM instance in order to define the real device type inside the FDT project. This device description assignment shall be done inside the engineering phase. There are two alternatives to do this:

- Use always a generic VAN Device DTM inside the project and assign a device description for an instance.
- Create a specific device type on the DTM type level by a symbolic assignment of a device description, before an instance of the VAN Device DTM will be placed in the FDT project.

For the integrated concept of the VAN Engineering Tool, the first alternative will be implemented. Therefore the DTM offers a special function called '*Configuration*', which can be used only once per DTM instance. After a successful device description assignment, this function will not further on offered by the VAN Device DTM.

6.2 VAN Communication DTM

In the previous deliverable D08.4-1 the FDT technology was introduced as a solution for a distributed and transparent automation system. Moreover, the architecture of the VAN Engineering Tool was provided. Inside this architecture, the VAN Communication DTM holds a relevant position. In the deliverable D08.4-1 the mandatory interfaces that a generic VAN Communication DTM has to provide were described. The scope of this chapter is to provide a more detailed description of all the interfaces required to enable the communication to a VAN device.

The VAN Communication DTM integrates a web service client to realize the calls to the web server of a VAN device. Furthermore, it has to provide standard FDT interfaces to communicate with the VAN Device DTM.

Figure 33 shows the main usage of the VAN Communication DTM. From one side it has to provide interfaces for the FDT/DTM technology and from the over side interfaces for the web services technology.

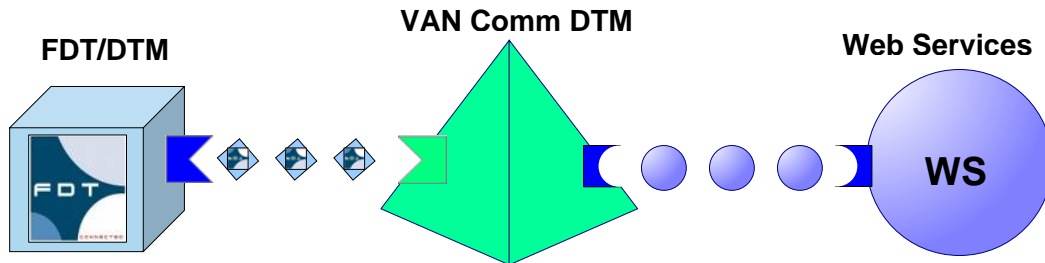


Figure 33: Role of the VAN Communication DTM

To recall the major points of the previous deliverable D08.4-1 the VAN Communication DTM has to realize a VAN customized Communication-channel, which allows to connect the VAN Device DTM and the VAN device. The objective of this communication is mainly the configuration of the ASE objects deployed inside the VAN device. The access to the ASEs will be done by means of web service technologies.

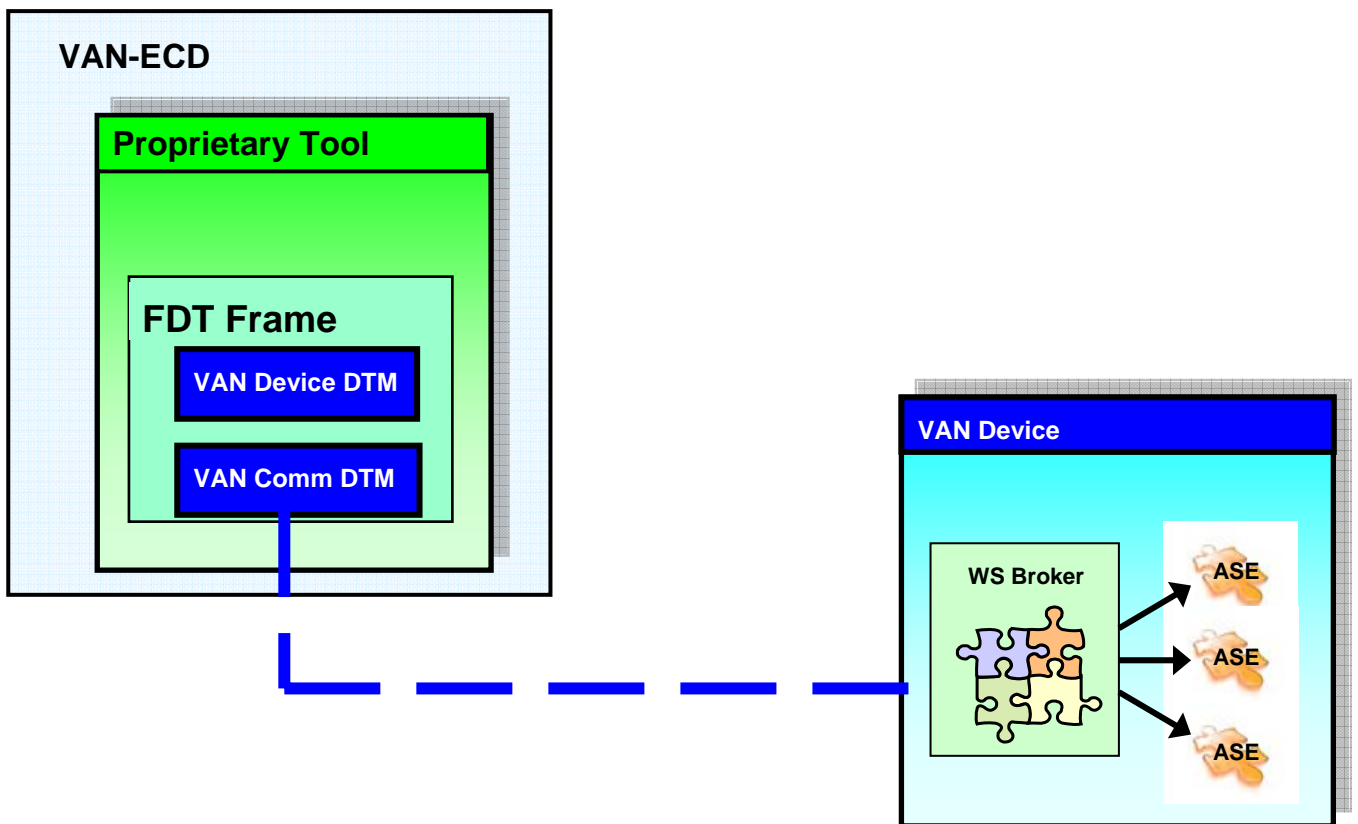


Figure 34: VAN Integrated Concept Architecture

Figure 34 shows the main architecture of the integrated concept with all its elements. In this chapter we will focus on the relations between VAN Device DTM, VAN Communication DTM, and the VAN device.

As shown in Figure 34 the VAN Communication DTM has to communicate with the Web Services Broker (WB Broker) to reach a specific ASE of the VAN device. The WS Broker is a software object in charge of the dispatching of the incoming calls to the right ASE.

An initial definition of ASEs was made in deliverable D02.2-2. In the first step of the development of a prototype for the integrated concept, two ASEs will be used to configure the VAN device, the *VAN Device Config* ASE and the *VAN Domain* ASE. Each ASE is mainly composed of two parts, its class specification, and its service specification. The class specification defines the attribute characteristics of the class. The service specification defines the services that are provided by the ASE [D02.2-2]. All the defined ASEs are using two generic services, *Get* and *Set*.

6.2.1 VAN Web Services and WSDL

As described in deliverable D02.2-2 web services are used to access the ASEs. They are a state of the art mean to communicate in heterogeneous network environments. They also permit to connect objects implemented on different platforms, and to provide a built-in approved security mechanism. Furthermore, web services allow communication across firewalls, which are required to guaranty the factory security, and they permit to connect different industrial sites in-between different companies.

The communication takes place within SOAP messages, which in VAN are transported on HTTP application layer, but other application layer transport protocols like FTP or SMTP are also useable.

The Communication DTM interacts directly with the WS Broker in the VAN device. It is the duty of the WS Broker to dispatch incoming calls to the correct ASE.

Figure 35 shows the concepts and provides an overview of how the communication between the VAN Engineering Client (VAN-EC) [D08.2-1] and the VAN device is realized.

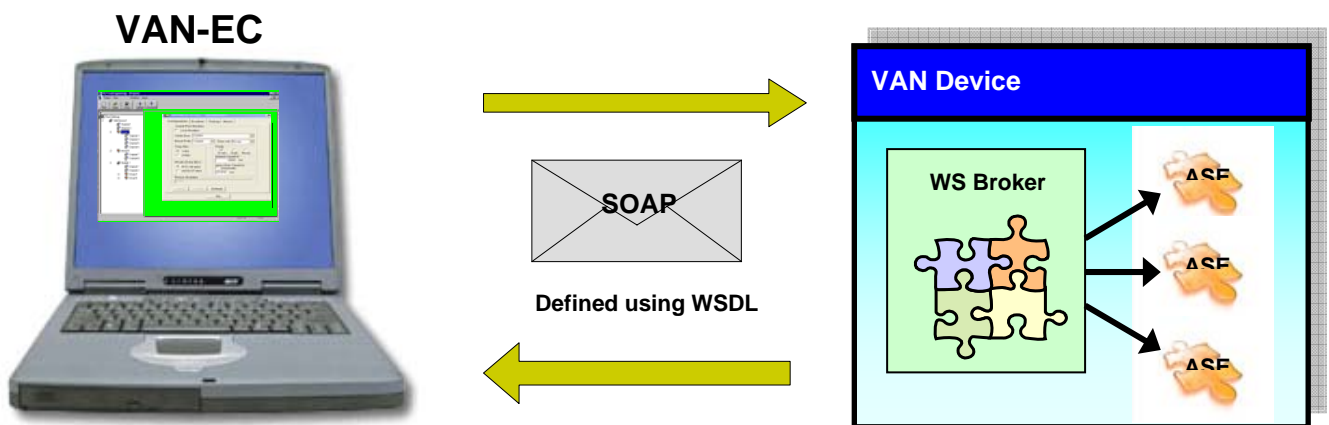


Figure 35 : Communication between VAN-EC and VAN Device

The starting point for the implementation of a web services application is to provide a WSDL document (Web Services Description Language), which describes how the communication between the client and server of a web service take place. Based on the WSDL file a web service client that realizes the defined functions can be generated. On the market free software tools are available that automatically generate the web service client from a WSDL file.

The WSDL file is a XML-base document, which provides a formal definition of the interface, so that requestor wishing to communicate with the service provider knows exactly how to structure the request message [SOA].

A WSDL file that represents the WS Broker for VAN devices will be provided from Tech PCC (Technical Project Coordination Committee) and it will be used to realize a first prototype of a VAN system. Since this generic WSDL is not focalized on the specific necessities of the VAN Communication DTM prototype, a reduced WSDL for the VAN Communication DTM based on the WS Broker is provided.

Figure 36 shows the structure of the WSDL file that will be used for the VAN Engineering Tool integrated concept. The *PortType* element represents a set of abstract operations. Each operation refers to an input message and output messages. The *Binding* element specifies concrete protocol and data format specifications for the operations and messages defined by a particular PortType. The *Service* element is used to aggregate a set of related ports.

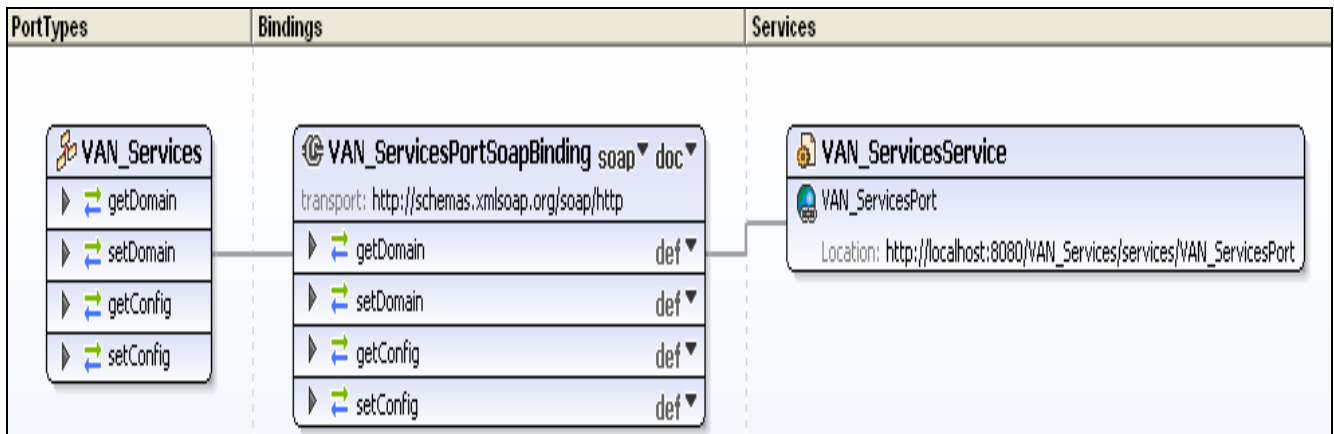


Figure 36: WSDL file for the VAN Prototype

6.2.2 SOAP in VAN

Simple Object Application Protocol (SOAP), this is the acronyms from one of the most quoted and affirmed application layer protocol for web services applications. It provides an XML messaging framework based on a client server architecture to exchange messages via remote procedure call.

Here is an example of how the VAN Communication DTM might create a SOAP message requesting the list of device configuration objects from a VAN Configuration ASE. The VAN Device DTM need to know the configuration objects instantiated in the VAN device.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getDeviceConfig xmlns="http://VAN-AD.example.com/ws">
      <list-of-device-config-objects >...</list-of-device-config-objects >
    </getDeviceConfig>
  </soap:Body>
</soap:Envelope>
```

And here is a possible response to the VAN Communication DTM request:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getDeviceConfigResponse xmlns="http://VAN-AD.example.com/ws">
      <getDeviceConfigResult>
        <list-of-device-config-objects >...</list-of-device-config-objects >
      </getDeviceConfigResult>
    </getDeviceConfigResponse>
  </soap:Body>
</soap:Envelope>
```

6.2.3 VAN Communication Schema

As already explained there are three main actors inside the FDT architecture: the FDT frame application, the FDT Device DTM and the FDT Communication DTM. Each one of these objects has to communicate with each other. The FDT communication schema represents the XML schema [W3ORG] used to standardize the communication between the VAN Device DTM and the VAN Communication DTM.

In the FDT domain, the communication schema is a way to standardize the messages exchanged between a device DTM and a communication DTM and it is strictly related to the fieldbus used to communicate to the device. It formalizes the types of data and the methods required to realize a data transmission on a specific protocol.

The particularity of the VAN solution is that for the configuration of the VAN parameters it is not used a specific fieldbus, like Modbus or Profinet. Instead, we are dealing with web services that are intrinsically a flexible solution and for this reason difficult to formalize in its whole integrity. This impossibility to completely describe the web service architecture itself lead us to focalize on a specific solution; in our case the web service client integrated in the VAN Communication DTM. Thus, all the methods used in the VAN web service client have to be formalized in the VAN Communication Schema

By the means of the VAN Communication Schema the VAN Device DTM and the VAN Communication DTM can parse the XML messages sent and received to evaluate correctness and semantics.

Figure 37 shows the architecture of the FDT communication from a VAN prospective. Both VAN Device DTM and VAN Communication DTM must provide standard FDT interfaces (blue ones) and specific VAN interfaces (green ones). The messages exchanged between the VAN Device DTM and the VAN Communication DTM are XML based and VAN Communication Schema compliant. The VAN Communication DTM interacts with the VAN device through the WS Broker using VAN web services client.

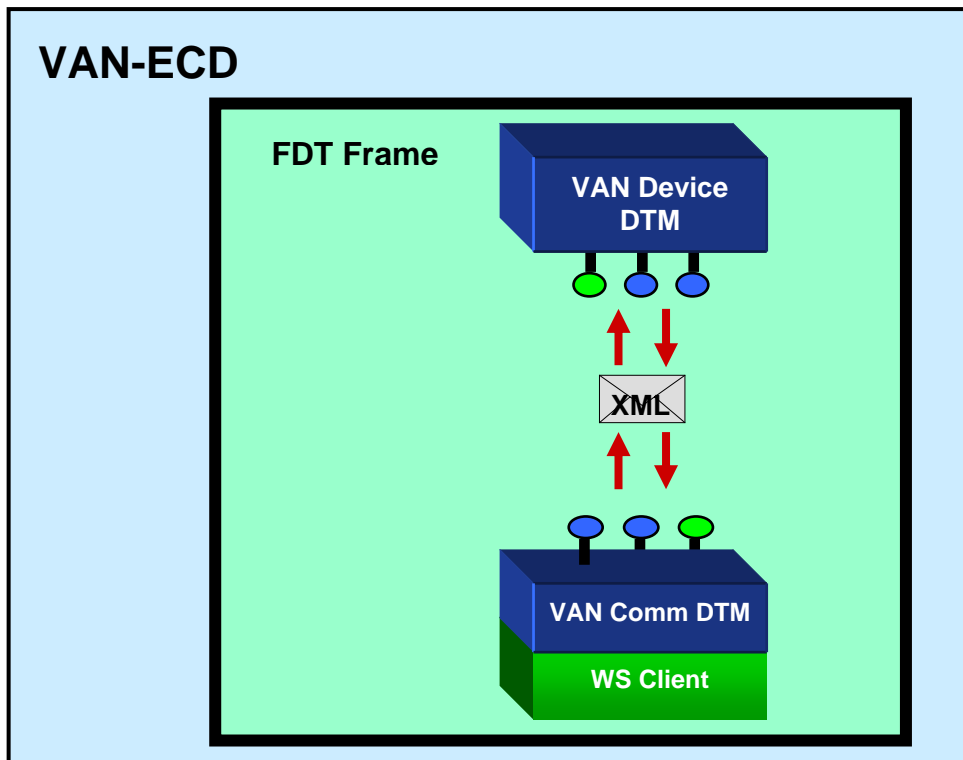


Figure 37: Architecture of the FDT Communication

Following, the VAN Communication Schema to be used for the implementation of VAN Device DTM and VAN Communication DTM is provided.

```
<?xml version="1.0" encoding="UTF-8"?>
<Schema name="VANCommunicationSchema"
  xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schemas-microsoft-com:datatypes"
  xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <AttributeType name="schemaVersion" dt:type="number" default="0.1"/>
  <AttributeType name="communicationReference" dt:type="uuid"/>
  <AttributeType name="objectReference" dt:type="string"/>
  <AttributeType name="xmlRequest" dt:type="string"/>
  <AttributeType name="xmlResponse" dt:type="string"/>
  <AttributeType name="deviceURL" dt:type="string"/>
  <!-- VAN DTM Transactions -->
  <!-- common for FDT -->
  <ElementType name="ConnectRequest" content="eltOnly" model="closed">
    <attribute type="deviceURL" required="yes"/>
  </ElementType>
  <ElementType name="ConnectResponse" content="empty" model="closed">
    <attribute type="communicationReference" required="yes"/>
  </ElementType>
  <ElementType name="DisconnectRequest" content="empty" model="closed">
    <attribute type="communicationReference" required="yes"/>
  </ElementType>
  <ElementType name="DisconnectResponse" content="empty" model="closed">
    <attribute type="communicationReference" required="yes"/>
  </ElementType>
  <!-- special for VAN -->
  <ElementType name="SetConfigReq" content="empty" model="closed">
    <attribute type="communicationReference" required="yes"/>
    <attribute type="objectReference" required="no"/>
    <attribute type="xmlRequest" required="no"/>
  </ElementType>
  <ElementType name="SetConfigRsp" content="empty" model="closed">
    <attribute type="communicationReference" required="yes"/>
    <attribute type="xmlResponse" required="no"/>
  </ElementType>
  <ElementType name="GetConfigReq" content="empty" model="closed">
    <attribute type="communicationReference" required="yes"/>
    <attribute type="objectReference" required="no"/>
    <attribute type="xmlRequest" required="no"/>
  </ElementType>
  <ElementType name="GetConfigRsp" content="empty" model="closed">
    <attribute type="communicationReference" required="yes"/>
    <attribute type="xmlResponse" required="no"/>
  </ElementType>
  <ElementType name="SetDomainReq" content="empty" model="closed">
    <attribute type="communicationReference" required="yes"/>
    <attribute type="objectReference" required="no"/>
    <attribute type="xmlRequest" required="no"/>
  </ElementType>
```

```

<ElementType name="SetDomainRsp" content="empty" model="closed">
  <attribute type="communicationReference" required="yes"/>
  <attribute type="xmlResponse" required="no"/>
</ElementType>
<ElementType name="GetDomainReq" content="empty" model="closed">
  <attribute type="communicationReference" required="yes"/>
  <attribute type="objectReference" required="no"/>
  <attribute type="xmlRequest" required="no"/>
</ElementType>
<ElementType name="GetDomainRsp" content="empty" model="closed">
  <attribute type="communicationReference" required="yes"/>
  <attribute type="xmlResponse" required="no"/>
</ElementType>
<!-- Main FDT element -->
<ElementType name="FDT" content="eltOnly" order="one" model="closed">
  <attribute type="schemaVersion"/>
  <attribute type="fdt:nodeId"/>
  <group order="one" maxOccurs="1" minOccurs="1">
    <element type="ConnectRequest"/>
    <element type="ConnectResponse"/>
    <element type="DisconnectRequest"/>
    <element type="DisconnectResponse"/>
    <element type="SetConfigReq"/>
    <element type="SetConfigRsp"/>
    <element type="GetConfigReq"/>
    <element type="GetConfigRsp"/>
    <element type="SetDomainReq"/>
    <element type="SetDomainRsp"/>
    <element type="GetDomainReq"/>
    <element type="GetDomainRsp"/>
    <element type="fdt:CommunicationError"/>
  </group>
</ElementType>
</Schema>

```

The VAN Communication Schema must define both the standard FDT interfaces and the specific VAN interfaces. The FDT interfaces are defined in the FDT specification. The VAN specific interfaces must represent methods (port type) and parameters used in the VAN Communication DTM according to the elements defined in Figure 36.

Based on the VAN Communication Schema all the XML files required for the communication between the VAN Device DTM and the VAN Communication DTM are generated. Following, an example of the *SetConfigReq* XML document is provided.

```

<?xml version="1.0" encoding="UTF-8"?>
<FDT xmlns="x-schema VANCommunicationSchema.xml"
  <SetConfigReq communicationReference="12345678-1234-1234-1234-123456789012"
    objectReference="thisIsARreference"
    xmlRequest="thisIsAXmlRequest"/>
</FDT>

```

The *communicationReference* is an FDT parameter that permits the VAN Communication DTM to identify a connection to a specific VAN Device DTM.

The *objectReference* is a mandatory key attribute that specifies a symbolic reference for the object instance.

The *xmlRequest* parameter is a XML file that should contain all the parameterization information on the required ASE.

6.2.4 VAN Communication DTM Sequence Diagram

To provide a better understanding of the phases of the communication between the VAN Device DTM and VAN Communication DTM a sequence diagram is provided. A sequence diagram describes an interaction between two or more objects focusing on the sequence of messages that are exchanged between them, and their corresponding occurrence on the lifelines [Wiki]. Table 15 describes the used elements.

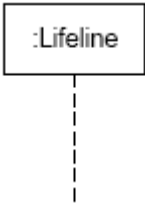
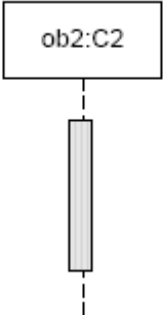
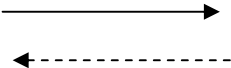
Node Type	Notation	Reference
Lifeline		A lifeline represents an individual participant in the interaction. Lifelines represent only one interacting entity.
ExecutionSpecification		An ExecutionSpecification is a specification of the execution of a unit of behaviour or action within the Lifeline.
Message		Messages come in different variants depending on what kind of Message they convey. Here we show a call and a reply.

Table 15: Sequence Diagram Notation

Sequence diagrams provide a visual representation of a use case. In our case, the use case analysed is the transmission of configuration data from the VAN Device DTM to the WS Broker. In the starting phase, we consider that the VAN Device DTM already has all the information required for the addressing and configuration of the VAN device. These data are partially added from the user and partially retrieved from the VAN Instance Model.

The VAN Communication DTM connects these two entities and manages the data transfer between them. The Figure 38 shows a sequence diagram where the VAN Device DTM goes online and transmits the configuration information to the VAN device. At the end, the communication is closed and the VAN Device DTM goes offline.

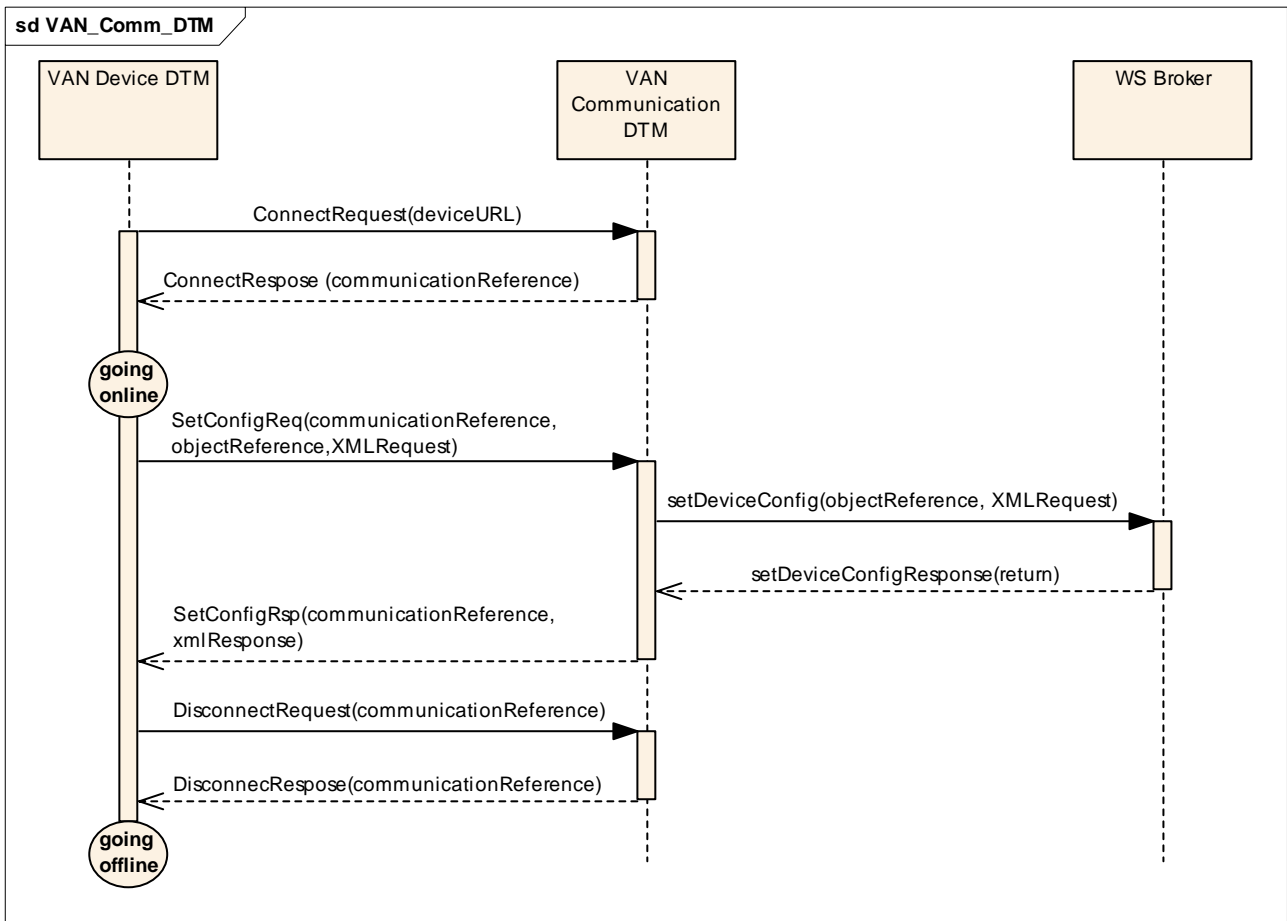


Figure 38: VAN Communication DTM Sequence Diagram

All the messages in Figure 38 exchanged between the VAN Device DTM and the VAN Communication DTM are consistent with the VAN Communication Schema defined in chapter 6.2.3. The messages exchanged between the VAN Communication DTM and the WS Broker are defined in the WS Broker’s WSDL file.

7 Conclusion

The scope of this deliverable was the development and refinement of the design and specifications for the object model and the interfaces, which are necessary for the engineering of VAN related attributes. The basic concepts and interfaces had already been identified in deliverable D08.4-1. These concepts were used in the deliverable at hand to elaborate the general specifications for the realization of necessary enhancements for VAN Engineering Tools.

The development of the specifications was based on a use case centric approach. Therefore, different use cases have been described. On the one hand, these use cases represent the view on a VAN system and the expected behaviour of a VAN Engineering Tool. On the other hand, the use cases are used to validate the design and specifications by concrete examples.

Although, the focus of deliverable D08.4-2 was on a generic specification for VAN Engineering Tools, especially the use case for the integrated concept was based on the user interface and features of the engineering tools Step7 and AutomationXplorer+. The decision to focus on existing tools was made because it is an inherent necessity of the integrated concept to cope with integration into existing tools and because the two mentioned tools will be extended by the project partners Siemens and Phoenix, respectively, to develop in the subsequent tasks of work package WP8 an engineering tool prototype for this concept. The detailed technical specifications for the implementation of a prototype of a VAN Engineering Tool following the integrated concept and for the integration in the existing engineering tools Step7 and AutomationXplorer+ needs to be elaborated in the first deliverable D08.6-1 of task T8.6.

For the stand-alone concept, the use case in this deliverable is generic and not biased by existing engineering tools. Parts of the VAN Engineering Tool following the stand-alone concept have already been implemented in a prototype of an Engineering GUI, which was successfully used for configuration of VAN devices in a first experimental setup of a VAN system. In addition, the use case and technical specifications for completion of a prototype for the stand-alone concept need to be detailed in the first deliverable D08.5-1 of task T8.5.

The VAN Information Model and corresponding VAN Instance Model as well as the VAN-DD and corresponding VAN-DD instance files are the common interface for all VAN Engineering Tools, both for the integrated concept and the stand-alone concept. Therefore, also the corresponding use cases and examples are relevant for both concepts and need to be considered for both tasks T8.5 and T8.6.

In the next tasks of work package WP8 the scope of the prototypes for VAN Engineering Tools has to be devised. The efforts need to concentrate on the requirements for the VAN Engineering Tools, which are necessary to support the prototypes of the different work packages and the experimental industrial setup in the VAN project. For that purpose, a questionnaire on the requested features and priorities had been send to all work package leaders and the outcomes have been discussed together with the Tech PCC. The first step in each of the tasks T8.5 and T8.6 is the elaboration of the technical specifications to a level, which is sufficient for implementation. After completion of the specifications in the first deliverable of each of the mentioned tasks, the implementation of prototypes for both concepts can start. The implementation of a prototype for the stand-alone concept will be the scope of the second deliverable D08.5-2 of task T8.5. The implementation of the different components and the integration into the existing engineering tools will be done in the second deliverable D08.6-2 of task T8.6.

To allow easy integration of emerging improvements of existing ASEs and newly developed ASEs, which will be defined by the other work packages of the VAN project, the design of the VAN-DD was based on a decomposed approach. Thereby, it is facilitated that the schema definitions of different ASEs and ASE classes can be managed in separate files, which in turn alleviates the maintenance and further development of the schemas by the responsible work packages.

Glossary

Terminology

ActiveX Control	Microsoft term that is used to denote reusable software components that are based on Microsoft Component Object Model (COM). ActiveX controls provide encapsulated reusable functionality to programs and they are typically but not always visual in nature.
Automation System	The complete automation system, which is to be engineered, i.e. the union of all VAN domains.
Configuration Data	Common database used by the VAN-ECDs during configuration, commissioning, production and maintenance. The configuration data contains all the information necessary to make the settings for each device in the automation system.
Communication Server	A communication component provided e.g. from the vendor of DP-Master / IO-Controller with a defined communication interface which allows the Device Tool to establish communication relation to a device [TCI].
Device	Any device used in an automation system, including network device.
Device Tool	In the TCI terms a Device Tool implements a graphical user interface optimized for the requirements of the corresponding device and transfers data from and to the device [TCI]. A Device Tool is an external program, which is accessible through TCI from the local engineering tool.
Engineering System	In the TCI terms the Engineering System is an engineering software used to engineer the complete system. The Engineering System calls the specific Device Tool to perform dedicated parameterization actions.
Hardware Catalogue	This is a catalogue of the local engineering tool, where all available objects for engineering are listed. The objects can be instantiated by drag and drop to the engineering window of the local engineering tool Simatic Manager.
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.
Module	Units to be used for the composition of a device.
Network Device	A special device, which controls the communication on a network segment (network card, modem, etc) or which interconnects multiple network segments (hub, switch, router, gateway, wireless access point, etc).
Network Segment	Physical grouping → Physical part of a communication network with its devices. See table 1.3 of [D02.2-1].

Transfermodule	Transfermodules define binary sequences of input or output ports of other devices. Each input or output port of any device is registered at a CPU under an address.
Transferunit	Is one or more bits as part of a transfermodule or the whole transfermodule
WS Broker	WS broker is the central WS server instance of a VAN device. It terminates the incoming WS call and checks for authorisation. Finally it distributes the call to the corresponding ASE.
VAN Device	A VAN device consists of one ore more application processes which define the device functionality and the connectivity to a VAN network. See chapter 2.2 of [D02.2-1].
VAN Device Description	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 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 parameterized 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-APs or VAN-ADs in the domain
VAN Instance Model	From the VAN Information Model it is possible to develop a VAN Instance Model that represents real use cases and customer requirements
VAN Information Model	The VAN Information Model represents the information necessary for the engineering of a VAN system. See chapter 2 of [D08.3-1].
VAN Network Segment	A part of a network which contains the VAN devices (VAN enabled or VAN aware). See table 1.3 of [D02.2-1].
XMI	The XML Metadata Interchange (XMI) is an Object Management Group standard for exchanging metadata information via Extensible Markup Language (XML). It can be used for any metadata whose metamodel can be expressed in Meta-Object Facility (MOF). The most common use of XMI is as an interchange format for UML models, although it can also be used for serialization of models of other languages (metamodels) [XMI].

Abbreviations

ACL	Access Control Layer
API	Application programming interface
ASE	Application Service Element
CA	Certification Authority
DD	Device Description
DHCP	Dynamic Host Configuration Protocol
DTM	Device Type Manager
FDT	Field Device Tool
FQDN	Full Qualified Domain Name
GSD	Generic Station Description
GSDML	General Station Description Markup Language
PID	Program Interface Description
PKI	Public Key Infrastructure
RA	Registration Authority
TCI	Tool Calling Interface
TPF	Temporary parameter file
UML	Unified Modeling Language
UUID	Universal Unique Identifier
VA	Validation Authority
VAN	Virtual Automation Network
VAN-AD	VAN Automation Device
VAN-AP	VAN Access Point
VAN-DD	VAN Device Description
VAN-EC	VAN Engineering Client
VAN-ECD	VAN Engineering Client for Automation Device
VAN-ECS	VAN Engineering Client for Automation System
WSDL	Web Service Description Language
WS	Web Services
XMI	XML Metadata Interchange
XML	Extensible Markup Language

References

- [D02.1-1] Deliverable D02.1-1 of the VAN project. *Report on selected basic technologies with deficiencies and proposal for VAN enhancements*. The VAN Consortium, 2005
- [D02.2-1] Deliverable D02.2-1 of the VAN project. *Topology Architecture for the VAN virtual Automation Domain*. The VAN Consortium, 2006
- [D02.2-2] Deliverable D02.2-2 of the VAN project. *VAN Open Platform API-Specification*. The VAN Consortium, 2006
- [D02.3-1] Deliverable D02.3-1 of the VAN project. *Application specific architecture for automation, VAN profile for process-manufacturing-industry*. The VAN Consortium, 2007
- [D06.1-1] Deliverable D06.1-1 of the VAN project. *Status and Analysis Report on security mechanisms and security infrastructures*. The VAN Consortium, 2006
- [D08.2-1] Deliverable D08.2-1 of the VAN project. *Engineering process concept and specification for the "VAN" platform*. The VAN Consortium, 2006
- [D08.3-1]. Deliverable D08.3.1 of the VAN project *Specification of product information model in general and of the mandatory product data*. The VAN Consortium, 2007.
- [D08.4-1] Deliverable D08.4-1 of the VAN project. *Specification of architecture and technologies for VAN engineering tool platform*. The VAN Consortium, 2007
- [FDT1.2.1] FDT-JIG: FDT-Joint Interest Group, *Guideline – FDT Interface Specification, V 1.2.1*. FDT-JIG - Order No. 0001-0001-002, 2005
- [FDT_G] FDT_JIG: FDT-Joint Interest Group *Guideline – Device Type Manager (DTM) Style Guide, V 1.0*, FDT-JIG - Order No. 0001-0008-000, 2005
- [IEC 61158] IEC 61158: Digital data communications for measurement and control - Fieldbus for use in industrial control systems Type 4 Profinet, Geneva 2003.
- [VAN Group] Homepage of the VAN Groupware: <https://www.van-eu.org/vangroupware>, The VAN Consortium, 2007
- [VANHP] Homepage of the VAN- Project: <https://www.van-eu.eu>, The VAN Consortium, 2007
- [SOA] *Service-Oriented Architecture Concept, Technology, and Design*. Thomas Erl, Prentice Hall, 2006.
- [TCI] PROFIBUS PROFINET Specification: *Tool Calling Interface*, Version 1.0, October 2006, Order No: 2.602
- [W3ORG] World Wide Web Consortium, <http://www.w3.org/XML/Schema>
- [Wiki] Wikipedia, www.wikipedia.org
- [WP2 API] Work Package 2, Open Platform and System Architecture: *API Manual*, The VAN Consortium, 2007