



**VAN**

**FP6/2004/IST/NMP/2 - 016696 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 D08.4-1

Specification of architecture and technologies  
for VAN engineering tool platform

<b>Document type</b>	: Deliverable
<b>Document version</b>	: Final
<b>Document Preparation Date</b>	: 02.07.2007
<b>Classification</b>	: Public
<b>Contract Start Date</b>	: 01.09.2005
<b>Duration</b>	: 31.08.2009



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

<b>Rev.</b>	<b>Content</b>	<b>Resp. Partner</b>	<b>Date</b>
0.1	Document structure and top-level activities	All	02.02.07
0.2	Architecture of VAN Engineering Tool	All	06.02.07
0.3	Updated contributions for top-level activities and technology introductions	All	16.02.07
0.4	Updated contributions for all technology introductions	All	06.03.07
0.5	Updated contributions for technology descriptions	All	27.04.07
0.6	Review final version	All	11.05.07
0.7	Incorporation of review results Executive Summary, Introduction and Conclusions	Schneider	21.06.07
1.0	Final modifications after review from task leader and work package leader	Schneider	02.07.07

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

## Executive summary

This document is the deliverable D08.4-1 of the VAN project and comprises the first 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 this task and indicates which of the different topics are covered by the two documents D08.4-1 and D08.4-2 to be delivered in the scope of this task. For the deliverable at hand, an overview of the content is provided and a short description is given for all relevant topics that will be further analysed inside the document. In particular, the reasons for development of two complementary concepts for the realization of a VAN Engineering Tool are given.

Chapter *Architecture for VAN Engineering Tool* recalls the role and tasks of the VAN Engineering Tool and its place inside the architecture of the complete VAN system. For the architecture of the VAN Engineering tool itself two different concepts are presented. The first concept targets the development of a stand-alone VAN Engineering Tool, which is independent from the existing engineering tools present on the market. The focus of the second concept is to facilitate the enhancement of existing engineering tools. This integrated concept is elaborated to allow the use of standardized technologies and interfaces as FDT/DTM and TCI. These two concepts will be used for the realization of first prototypes of VAN Engineering Tools. Therefore, a simplified use case of a VAN system is presented, in order to derive a VAN Instance Model of reasonable size and complexity. This instance model is used to validate the architecture and interfaces of the two concepts by specification of two concrete realizations of VAN Engineering Tool prototypes. Based on the prototype for the integrated concept also the common data exchanged between the tools is analyzed.

Chapter *VAN Device Description (VAN-DD)* clarifies where the VAN-DD is used in the VAN Engineering Tool architecture for the integrated concept as well as for the stand-alone concept. Furthermore, requirements regarding the VAN-DD coming from the other work packages, mainly from WP2, are summarized and the process for the generation of configuration data is explained.

The next two chapters provide basic explanation of the technologies used for the integrated concept. Both chapters only describe those aspects, which are relevant for VAN.

The *Tool Calling Interface (TCI)* chapter gives an overview of the TCI technology, which permits to call one software tool from another tool and to pass parameters between these tools. In the integrated concept TCI is used to couple Step7 and AutomationXplorer+. Furthermore, this chapter analyzes the adaptations for a seamless integration of this technology in the existing tools, which should be enhanced for the engineering of VAN systems.

In the same way, the *FDT for VAN-Engineering* chapter provides a description of all the elements of this standard, which are relevant for VAN. Here the VAN Device DTM and VAN Communication DTM are introduced. These two elements are used by the AutomationXplorer+ in order to realize the communication to the VAN device and to perform the configuration of the VAN related attributes of these devices.

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

<b>1 Introduction</b>	<b>6</b>
<b>2 Architecture for VAN Engineering Tool</b>	<b>8</b>
2.1 Engineering Workflow	10
2.2 Concepts and Solutions for VAN Engineering Tools	12
2.2.1 Stand-Alone Concept	12
2.2.2 Integrated Concept	14
2.3 Approach for an VAN Instance Model	17
2.4 Realization of a VAN Engineering Tool Prototype	19
2.4.1 Prototype for Stand-Alone Concept	19
2.4.2 Prototype for Integrated Concept	22
2.4.2.1 Introduction to Step7	23
2.4.2.2 Introduction to AutomationXplorer+	25
2.5 Common Data	27
<b>3 VAN Device Description (VAN-DD)</b>	<b>32</b>
3.1 Introduction	32
3.2 Use of a Device Description within the Integrated and Stand-Alone Concept	33
3.3 Basic Requirements on VAN-DD	35
3.4 Concept of the VAN-DD	35
3.5 Workflow of Data Generation	38
<b>4 Tool Calling Interface (TCI)</b>	<b>39</b>
4.1 Introduction to TCI	39
4.2 Program Interface Description (PID)	41
4.3 Temporary Parameter File (TPF)	42
<b>5 FDT for VAN-Engineering</b>	<b>43</b>
5.1 Introduction to FDT/DTM	43
5.2 Tasks for the FDT Components in VAN	44
5.3 VAN Device DTM	47
5.3.1 Internal Architecture of a VAN Device DTM	47
5.3.2 Adaptation of the VAN Device DTM to the VAN Device	48
5.4 VAN Communication DTM	50
5.4.1 Communication-channel Object Model	51
5.4.2 Communication DTM Interfaces	52
5.4.3 Roadmap for Developing a VAN Communication DTM	54
<b>Conclusion</b>	<b>55</b>
<b>Glossary</b>	<b>56</b>
Terminology	56
Abbreviations	58
<b>References</b>	<b>59</b>

## List of figures

Figure 1: Refined Engineering Tool Integration in VAN System .....	8
Figure 2: Communication between Engineering Tools and Devices.....	9
Figure 3: Refined Engineering Process.....	11
Figure 4: VAN Engineering Tool Concepts .....	12
Figure 5: Stand-Alone Tool within VAN Architecture.....	13
Figure 6: Complete Integration of VAN Enhancements .....	14
Figure 7: Integration of VAN Enhancements with FDT/DTM.....	15
Figure 8: Integration of VAN Enhancement with TCI and FDT/DTM.....	16
Figure 9: Example of a Factory Automation Topology .....	17
Figure 10: Reduced Example of a Factory Automation Topology.....	18
Figure 11: First Outline of Tool Components .....	20
Figure 12: Schema of a Stand-Alone Engineering Solution.....	21
Figure 13: Configuration of VAN Devices Using TCI and FDT Frame Application.....	22
Figure 14: User Interface of Step7 .....	24
Figure 15: Structure of the Temporary Parameter File.....	31
Figure 16: Correlation of a DD to the Engineering Process .....	32
Figure 17: Use of a DD for the Integrated Prototype.....	33
Figure 18: Use of a DD in the Stand-Alone Prototype.....	34
Figure 19: VAN-DD Overview .....	36
Figure 20: Generic Structure of an ASE.....	37
Figure 21: Invocation Interface – Overview.....	39
Figure 22: Communication Server Used by the Device Tool .....	40
Figure 23: Structure of a PID File.....	41
Figure 24: FDT Architecture .....	44
Figure 25: Requirements for a VAN Device DTM .....	47
Figure 26: Possible Internal Structure of a VAN Device DTM.....	47
Figure 27: Information Flow from Engineering Tool to FDT Frame Application .....	48
Figure 28: Generation of a DTM based on a VAN-DD .....	48
Figure 29: Basic Communication Interfaces.....	50
Figure 30: FDT Object Model FdtChannel .....	52

# 1 Introduction

The objective of task T8.4 is to develop a concept for the engineering of VAN related attributes in a VAN system and to specify the interfaces used to integrate the necessary enhancements for VAN in engineering tools. Since the overall task runs for 14 months it was decided to release two deliverables. The focus of this first deliverable D08.4-1 is to describe the architecture of a VAN Engineering Tool and to identify the interfaces and technologies used for the proposed concepts. The results of this deliverable are synchronized with the other work packages before the detailed technical specification of the object model and interfaces will be developed for the second deliverable D08.4-2.

Before starting the description of the architecture and the specification of the interfaces, it is important to clarify and understand the context and the behaviour of the VAN Engineering Tool inside the complete VAN system architecture. One important aspect to keep in mind is that the VAN Engineering Tool is not an integral part of the VAN system itself, but it is an external actor connected to the VAN network only during the engineering phases. Additionally, it was already defined in deliverable [D08.3-1] that the VAN Engineering Tool has to cover two main roles: the VAN-ECS for planning of the complete VAN system and the VAN-ECD for configuration and parameterization of a VAN device.

Since most of the existing engineering tools are limited to a scope, that corresponds to the role of a VAN-ECD, two different concepts are presented in this deliverable:

On the one hand a stand-alone concept is proposed, which can be used for development of new tools, which focus only on VAN aspects and which cover both roles of the VAN-ECS and VAN-ECD. Since such stand-alone tools focus on the VAN aspects, they should natively support all enhancements necessary for the engineering of a VAN system and the communication to VAN devices. The user interface and the user guidance of stand-alone tools can be designed with regard to the needs of a VAN system. One major advantage of stand-alone VAN Engineering Tools is that they are independent from existing engineering tools used for other engineering tasks of automation systems. Therefore, no changes are necessary for existing engineering tools and they can be supplemented by the stand-alone VAN Engineering Tools for the engineering of the overall VAN system.

On the other hand an integrated concept is presented, which targets on the integration of the necessary VAN enhancements into existing engineering tools. The main advantage is that the user needs to work with only one tool to perform all engineering tasks. However, since most of the existing engineering tools do not cover the scope of a VAN-ECS the integrated concept does focus only on the functions of the VAN-ECD role. The straightforward approach for an integrated concept is to add all necessary extensions for VAN directly in the implementation of an existing engineering tool. This approach implies that each engineering tool has to be modified. In order to facilitate the integration of VAN specific features one goal of work package WP8 is to provide reusable software components. One possible solution for that task is to provide source code of the implementation of the VAN enhancements. This source code still needs to be integrated and adapted to the implementation and the user interface of an existing engineering tool. Another approach is to use standardized interfaces to integrate compiled and tested libraries into an existing tool. A public available specification for that purpose is the FDT/DTM technology. Based on this technology, DTMs can be provided, which support the engineering of VAN devices and the communication from the engineering tool to the VAN devices across heterogeneous networks. These DTMs can be used by any existing engineering tool, which supports the FDT technology. Moreover, it is also possible for existing engineering tools, which do not directly support the FDT technology, to exploit the DTMs, if they are able to call other tools via the standardized TCI interface. These tools can call another tool, which is able to receive a TCI call and which supports the FDT technology and thus can use the DTMs. The advantage of using established and public available standards for the integrated concept is a smooth integration in the existing automation tool market, where automation solution providers can enhance their existing products and at the same time protect the investments already made.

The feasibility of both the stand-alone concept and the integrated concept has been investigated by definition of concrete implementations for VAN Engineering Tool prototypes. For the stand-alone concept a component-based design for an Engineering GUI is presented, which can be realized based on the Eclipse platform [ECLIPSE]. The Engineering GUI is structured in an overview level, which focuses on the functions necessary for the VAN-ECS role, and a configuration level, which implements the VAN-ECD features. The components and interfaces of the Engineering GUI are also designed in a way, which allows to reuse parts of the implementation in the prototype or DTMs for the integrated concept. For the prototype of the integrated concept the examined realization is based on the tools Step7 from Siemens and AutomationXplorer+ from Phoenix. These tools already implement the TCI and FTD/DTM technology, respectively, and since the suppliers of both tools are participants of work package WP8 it is possible to implement missing functions and features. Step7 can be enhanced to represent VAN Network Segments along with attached VAN devices and it can make a TCI call to AutomationXplorer+ for parameterization of a VAN device. AutomationXplorer+ receives the TCI call and instantiates the DTMs for configuration of the VAN specific parameters and download of the configuration to then VAN device.

Independent of the selected concept and technologies any VAN Engineering Tool must be able to exchange information with the VAN Instance Model, which describes the complete VAN system but also holds the parameters and actual values for the individual instances of the VAN devices. To identify and validate the interface between the VAN Engineering Tool and the VAN Instance Model a simple use case of a VAN system is required. In the deliverable [D02.2-3] a use case for factory automation was already defined. Nevertheless, the complexity of this use case is still too high to be used in the first step. For this reason in this deliverable a simplified version is defined. This simplified use case can also be used to provide a first implementation of the VAN Instance Model.

Furthermore, any VAN Engineering Tool must be able to read a VAN-DD. It is important to define where and when the VAN-DD is used inside the proposed concepts. Moreover, the identification of the key information that the VAN-DD must contain takes a significant importance in the specification process. Since the VAN-DD represents a VAN device, it must be able to describe the parameters of all ASEs implemented by the VAN devices, as they are formally specified in the deliverable [D02.2-2].

## 2 Architecture for VAN Engineering Tool

This chapter provides an overview of the VAN architecture from the VAN Engineering Tool point of view. Starting from the VAN engineering concept the whole architecture will be explained and possible solutions will be provided.

It is important to clarify where and when the VAN Engineering Tool is used inside the VAN environment. The VAN Engineering Tool is not a part of the VAN domain itself see Figure 1. The VAN Engineering Tool is a software application used during the engineering phases of the VAN system. It can cover the role of the VAN Engineering Client for Automation System (VAN-ECS) or VAN Engineering Client for Automation Device (VAN-ECD) or both (see [D08.2-1]).

The automation applications required by the system are engineered using existing proprietary tools. It is not in the focus of work package WP8 to modify or provide new automation application solutions. Instead, it has to support the parameterization of the new VAN common Application Service Elements (ASE) [D02.2-1] and just those.

The VAN Engineering Tool as already said is not an integrated part of the VAN system. Therefore it is not always directly connected to the VAN system. It is plausible to suppose that it runs on a workstation or a PC and, on demand, it is connected to the VAN system in order to perform network management actions or VAN device configuration.

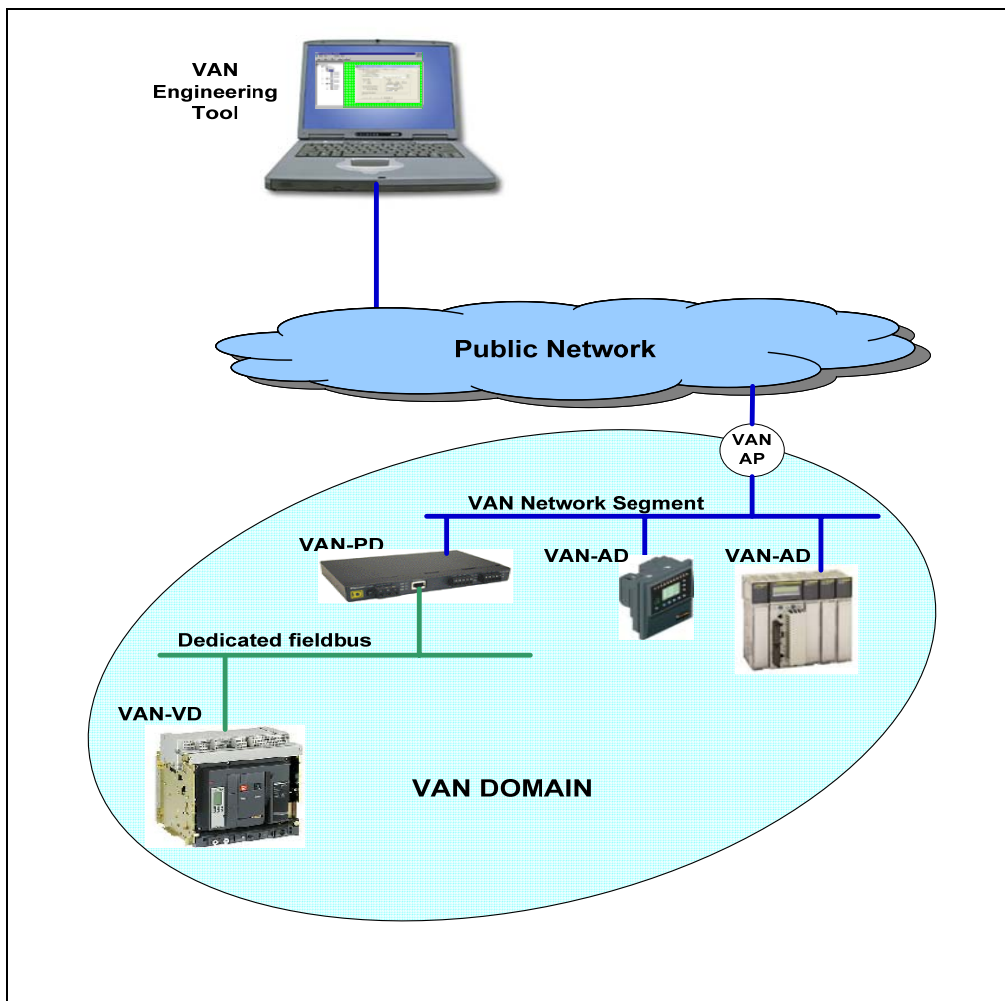


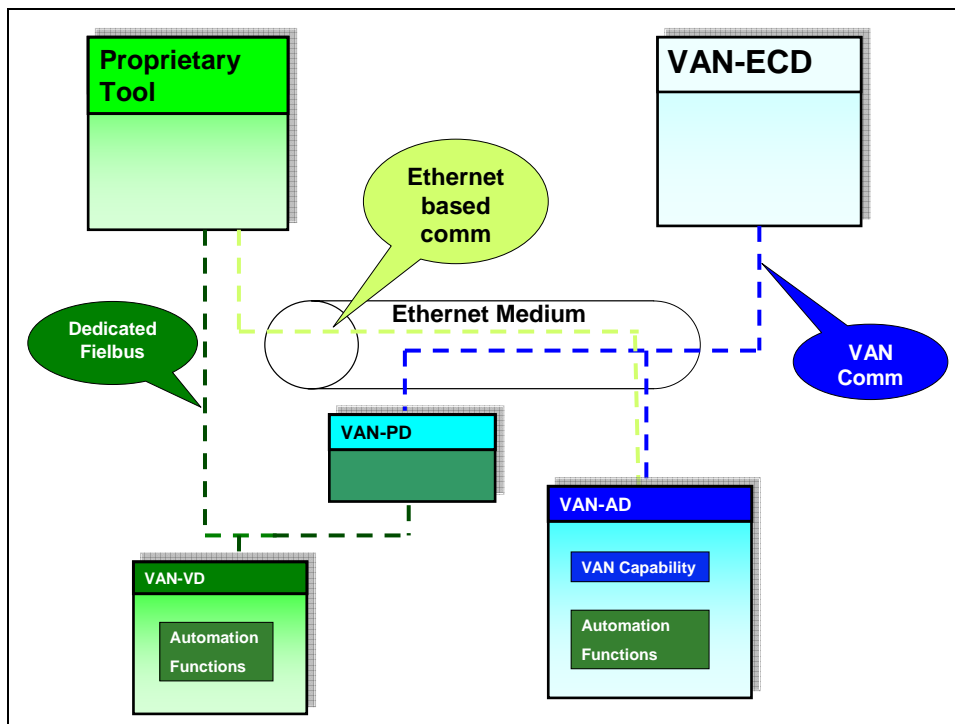
Figure 1: Refined Engineering Tool Integration in VAN System

Figure 2 shows an overview of a possible configuration of a VAN System, where a VAN Automation Device (VAN-AD), a VAN Proxy Device (VAN-PD) and a VAN Virtual Device (VAN-VD) are presented. All these elements are defined in the deliverable [D02.2-1].

“A VAN Automation Device defines an entity containing VAN services and protocol implementation and does contain at least one automation function or one automation application process in the VAN application context”. [Chapter 1.2.3.5 of D02.2-1]

“A VAN Proxy Device defines an entity containing VAN Function Set (VAN-FS) and Proxy Application. It is a proxy to automation devices (VAN Virtual Devices) in a VAN application context (VAN domain) which can be accessed within the VAN domain name space”. [Chapter 1.2.3.4 of D02.2-1]

“A VAN Virtual Device defines an entity without VAN network capabilities but with at least one automation function or one automation application process in the VAN application context. These devices are members in a VAN domain and will be accessed e.g. via a VAN Access Point”. [Chapter 1.2.3.6 of D02.2-1]



**Figure 2: Communication between Engineering Tools and Devices**

In Figure 2 the proprietary tool and the VAN-ECD are shown as separate entities in order to show the two different communication channels of the architecture.

The final VAN system should be a mix between the existing solution and the new VAN features, to provide an easy and reliable integration in the existing automation systems.

Nowadays, automation existing engineering tools and devices typically compose automation systems, which are connected by dedicated fieldbus technologies. Here the proprietary tool is directly connected to his device by a dedicated fieldbus.

It is important to remark that a VAN device is an existing device enhanced to integrate some VAN capabilities. For this reason if a proprietary tool is able to use an Ethernet based communication it can access to a VAN device directly to provide the automation application required from the system. A proprietary tool as Step7 (programming software by Siemens) can be connected to a Siemens PLC by a Profinet fieldbus and download the configuration for the automation applications required in the system. If the Siemens PLC integrates as well VAN functionalities, the configuration of those VAN functionalities are provided from the VAN-ECD. In this case, the VAN device is configured from two different channels, the VAN communication channel and the dedicated communication channel, both using Ethernet as a physical medium, as shown in Figure 2.

The VAN-AD is connected to one physical network, the Ethernet medium. But it is accessible from two logical communication channels: one based on VAN technology and one based on dedicated technology for example Profinet.

The VAN Virtual Device defines an entity without VAN network capabilities but with at least one automation function or one automation application process in the VAN application context [D02.2-1]. This device is reached from the VAN-ECD through a VAN-PD.

## 2.1 Engineering Workflow

An important aspect to be addressed is whether the engineering tool should cover all the phases of the engineering process or not. The engineering process of a VAN system was already explained in the deliverable [D08.2-1], where all the phases were analysed in detail. In the same deliverable the VAN Engineering Client for Automation Device and VAN Engineering Client for Automation System were introduced to define two distinct roles that the engineering tool can cover during the engineering workflow process.

“The VAN-ECS is used for engineering of the complete automation system. The VAN-ECS does not know all details on devices and configuration and it is used mainly for modelling and planning the complete automation system” [Chapter 1.1.3. of D08.2-1].

The VAN-ECS is used to execute all tasks, which are necessary during modelling and planning phases. Commercially available UML (Unified Modelling Language) tools can realize the VAN-ECS.

“The VAN-ECD is used for engineering of a single VAN device. The VAN-ECD does not have the information about the complete automation system, but it has detailed information for the automation component to be engineered and the communication relations of this automation component. A VAN-ECD is used mainly for configuration and commissioning and it is typically covered by some vendor specific engineering tool” [Chapter 1.1.3. of D08.2-1].

In relation to the generic workflow introduced in [D08.2-1] the following refinements have been added (see Figure 3). The engineering workflow is split in two parts, which are covered by the VAN Engineering Tool. The direct link between planning and configuration has been removed. The connection is now re-established via the VAN Instance Model. The reason for this step is the higher level of views of a VAN system. The first steps are holistic steps and the engineered data will be accessible by two different local engineering tools. For that, the first phases are realized by a VAN-ECS. A VAN-ECS can be a separate tool or it can be integrated within an existing engineering tool. The modelled and planned data will be stored in a VAN Instance Model, which is accessible for further use during the engineering process. The modelled and planned data can be exported and imported from/to the VAN-ECS with the XMI “XML Metadata Interchange format”. The next steps, starting with configuration, will be realized with the VAN-ECD.

As shown in Figure 3 the VAN-ECS and VAN-ECD are used to perform two different parts of the engineering process. The VAN-ECS is responsible for the modelling and planning phases and the VAN-ECD is responsible for the configuration, commissioning and production phases.

An important component of the VAN engineering workflow is the VAN Information Model. The VAN Information Model was introduced in the deliverable [D08.2-1]. Within the Instance Model the complete architecture of a VAN system from the engineering point of view is represented. It is a helpful base for design during the first engineering steps.

The VAN Instance Model contains all data that represents the whole VAN system. It contains all the information required to parameterise the VAN devices. The VAN-ECS is used to generate the VAN Instance Model of the VAN system. The second part of the engineering process, covered by the VAN-ECD, accesses the VAN Instance Model and uses this information to realize the configuration, commissioning and production phases for a specific VAN component.

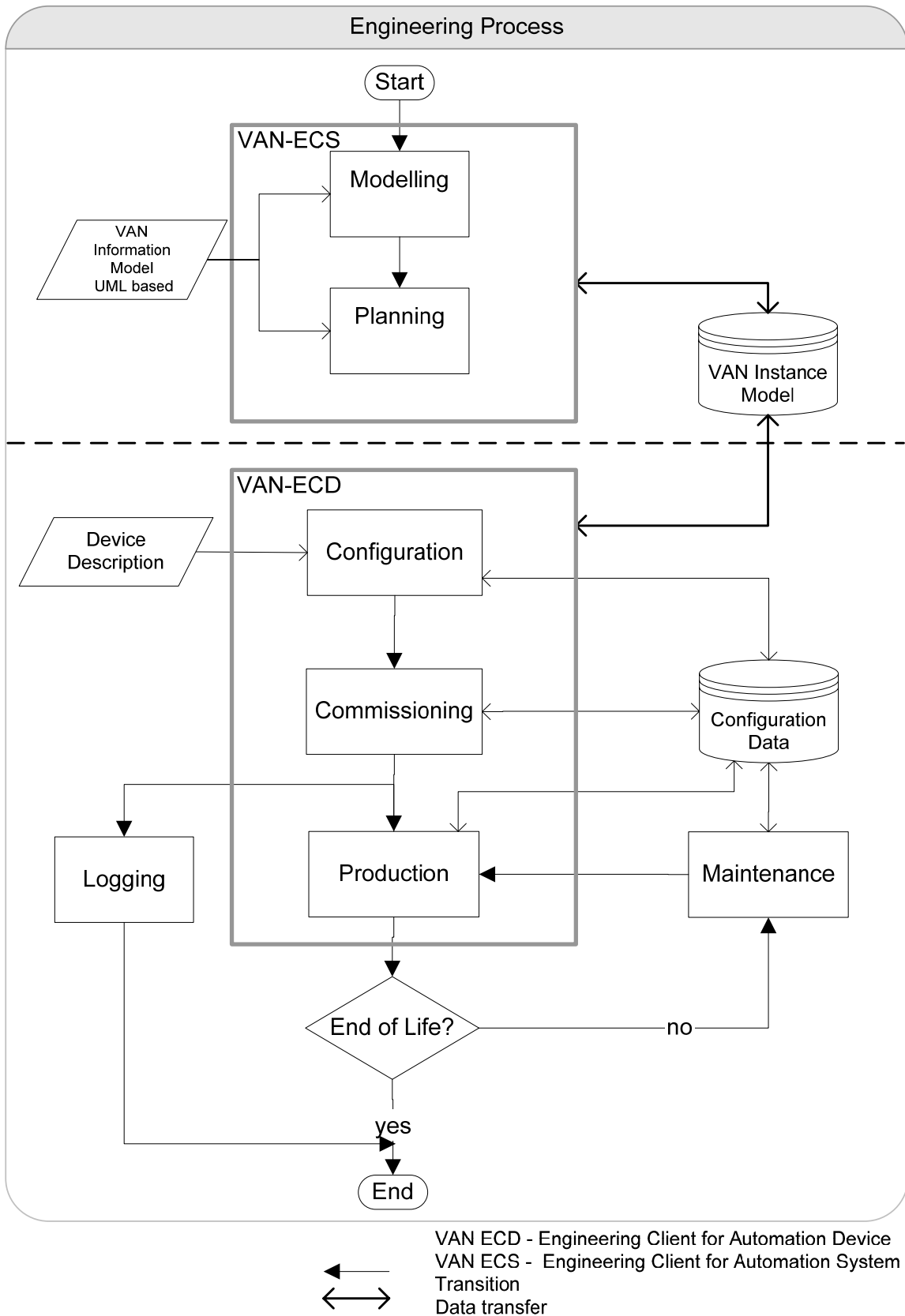


Figure 3: Refined Engineering Process

## 2.2 Concepts and Solutions for VAN Engineering Tools

Figure 4 shows two different possible concepts to realize a VAN Engineering Tool. One where the new VAN enhancements are completely integrated in the existing proprietary tool and the other where a stand-alone tool provides these enhancements. In the following paragraphs, both solutions will be investigated and analysed.

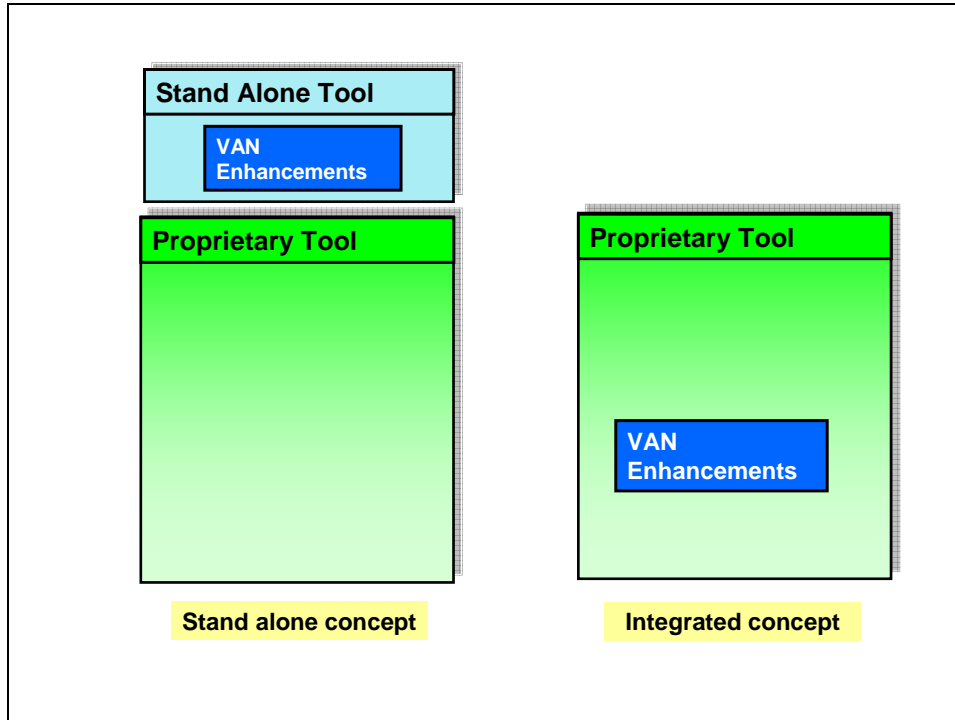


Figure 4: VAN Engineering Tool Concepts

### 2.2.1 Stand-Alone Concept

The following paragraph describes a possible stand-alone solution to fulfil different engineering tasks during the VAN engineering process.

The goal of a stand-alone solution is to provide a user interface easy to handle for modelling and planning of system. This task covers the defined VAN-ECS role. A further task of such a tool should be to parameterize the device by means of ASEs. This includes a mechanism to read/write data from/to the devices via web services. This task is part of the defined VAN-ECD role. A component-based tool with a graphical user interface (GUI) could cover these roles.

The benefit of such a stand-alone solution is that no special effort within each locally used engineering tool is required. Only a compatible link must exist. This link between a local engineering tool and a stand-alone solution in the VAN context is the VAN Instance Model. If a stand-alone solution is able to exchange data with the VAN Instance Model, it is thus independent from proprietary engineering tools and can be used by a variety of customers and solution providers.

Figure 5 shows how such kind of tool is embedded in the engineering architecture of VAN.

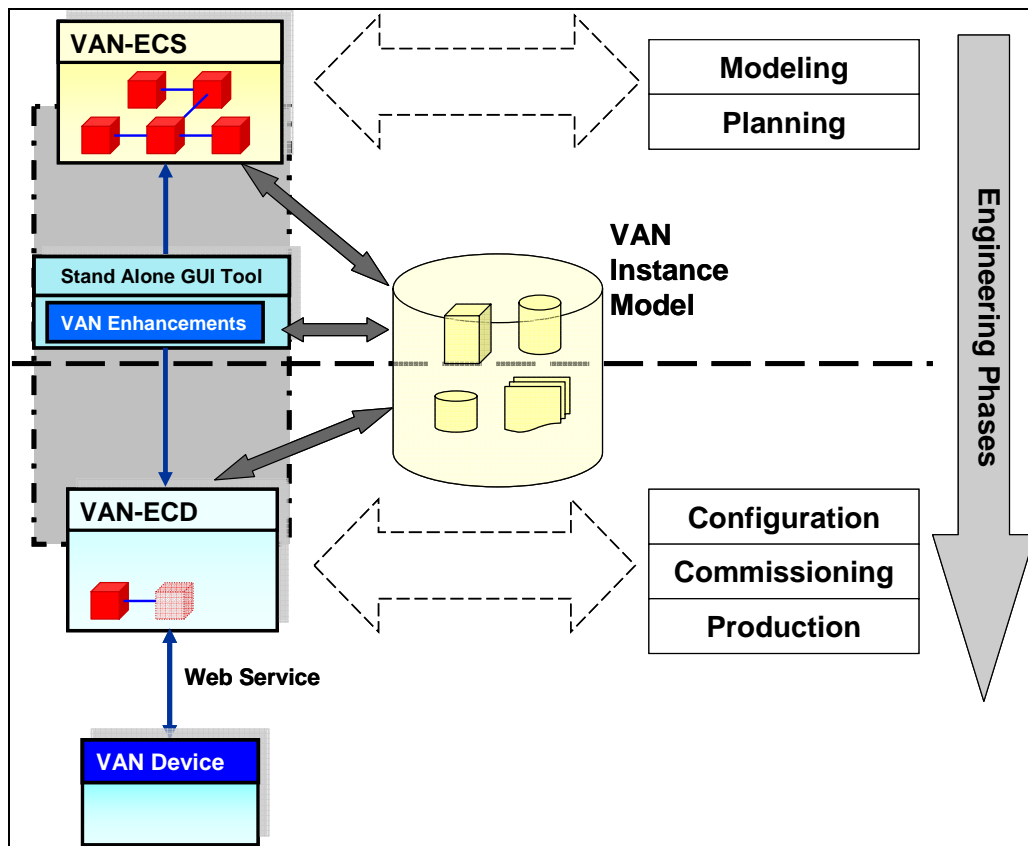


Figure 5: Stand-Alone Tool within VAN Architecture

An engineering tool with a graphical user interface should be designed to follow the top-down approach. Different hierarchical levels, each with different functions are useful to realize the top-down approach. The first level (overview) is used to define the VAN domain and its subdomains and logical relations. The needed information comes from the engineer himself and also from the already available UML based VAN Instance Model. The open platform allows, with its interfaces, on one side to design (parts of) the VAN Instance Model and on the other side the use of an available VAN Instance Model for further work. The second level (configuration) represents components in one VAN subdomain. Besides, it is also possible to engineer several subdomains in an own on register and graphical interpretation view. For configuration of VAN devices, the tool has to parameterize the VAN devices by means of ASEs. A register with a form, which can be used to fill in the needed parameters for an ASE of a given device, must be available for the responsible engineer. Before the development of such an engineering tool, the interfaces to the environment and to other tools have to be defined and specified. In chapter 2.4, the approach to design such a prototype is described more in detail.

### 2.2.2 Integrated Concept

To integrate the VAN enhancements in existing proprietary tools several approaches are possible. The first approach (see Figure 6) consists of a complete integration of the VAN functionality inside the existing tool. This means that the enhanced existing tool has to provide access to the ASEs and as well it must be able to access the VAN Instance Model.

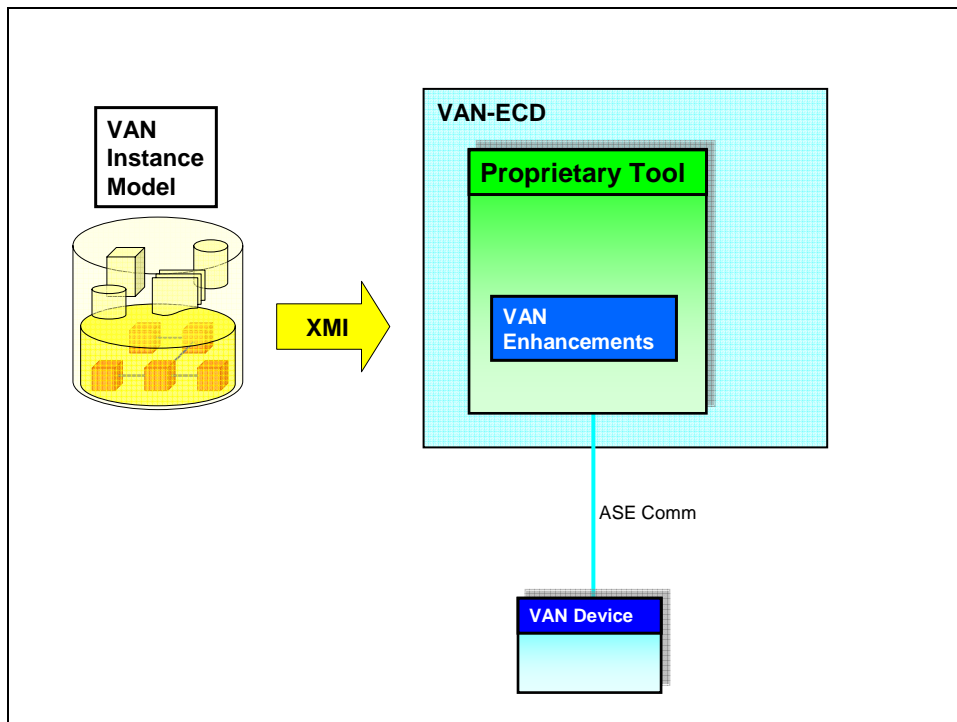


Figure 6: Complete Integration of VAN Enhancements

The second approach is used for proprietary tools that already have a FDT Frame Application (see chapter 5) integrated. Following this approach a VAN Device DTM (see chapter 5) and a VAN Communication DTM have to be provided, which are able to parameterize the ASE (see Figure 7). This VAN Communication DTM and the VAN Device DTM can be easily reused in every FDT Frame Application.

The existing tool must be able to import and export all the VAN configuration information from the VAN Instance Model. The proprietary tool accesses the VAN Instance Model to get an overview of the topology and structure of the VAN system. For a specific VAN device the FDT Frame Application has to instantiate the corresponding VAN Device DTM. This Device DTM is taking care of the parameterization required for a specific VAN device. To perform this task the Device DTM must be able to find the information for the specific device inside the VAN Instance Model.

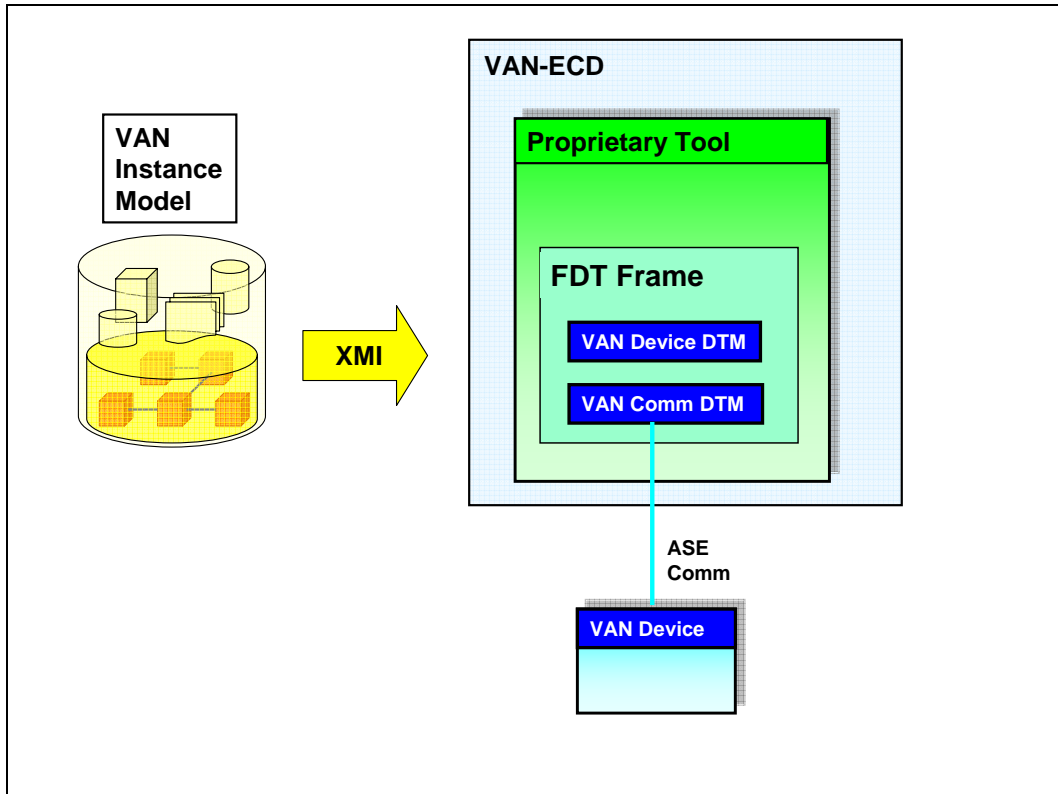


Figure 7: Integration of VAN Enhancements with FDT/DTM

Proprietary tools with no FDT Frame Application integrated can use a separate FDT Frame Application. In this case, the FDT Frame Application is not integrated in the tool but is a separate application see Figure 8.

the proprietary too access to the VAN Instance Model and using the TCI technology (see chapter 4) calls a separate FDT Frame Application and pass to it the information required to identify the device. Is an issue of the FDT Frame Application to instantiates the required VAN Device DTM and the VAN Communication DTM. Detailed information needed for the parameterization of the VAN devices are retrieved by the Device DTM directly from the VAN Instance Model.

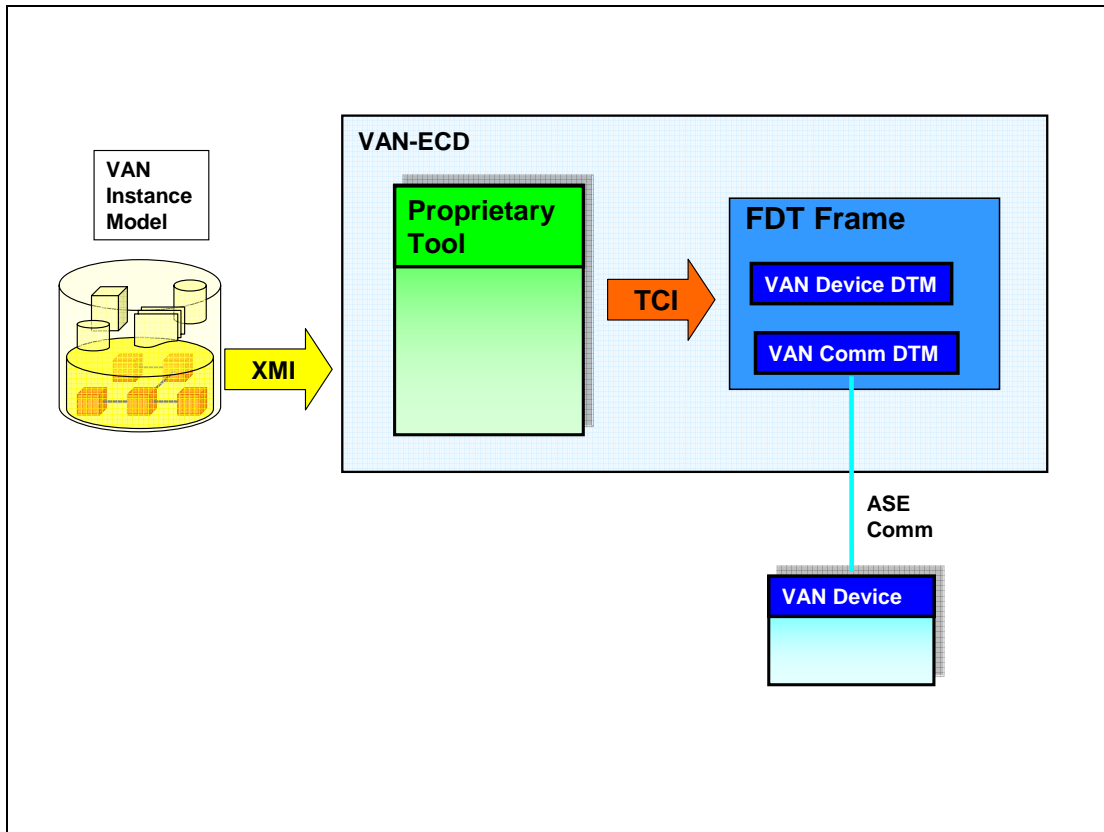


Figure 8: Integration of VAN Enhancement with TCI and FDT/DTM

### 2.3 Approach for an VAN Instance Model

Based on the instantiation example introduced in Figure 1.10 of [D08.2-1], the following Figure 9 shows an enhanced network topology. The background of this topology is the use case for factory automation defined in [D02.2-3]. The detailed description of the duties and responsibilities of this use case is also described in [D02.2-3]. Derived from this use case and descriptions made in task T8.2, parts of it are considered and merged together to create one practical use case. This use case serves as platform for VAN engineering to develop a VAN Instance Model.

The use case consists of two plants connected via public network. Within each plant are machines and robots. To fulfil their task the machines and robots are connected via LAN. For assembling the resulting products, the two plants must be synchronised. This use case allows a mix up of currently used devices with VAN devices, which will be the most typical configuration used in industry in the first phase of spreading of the VAN standard [D02.2-3].

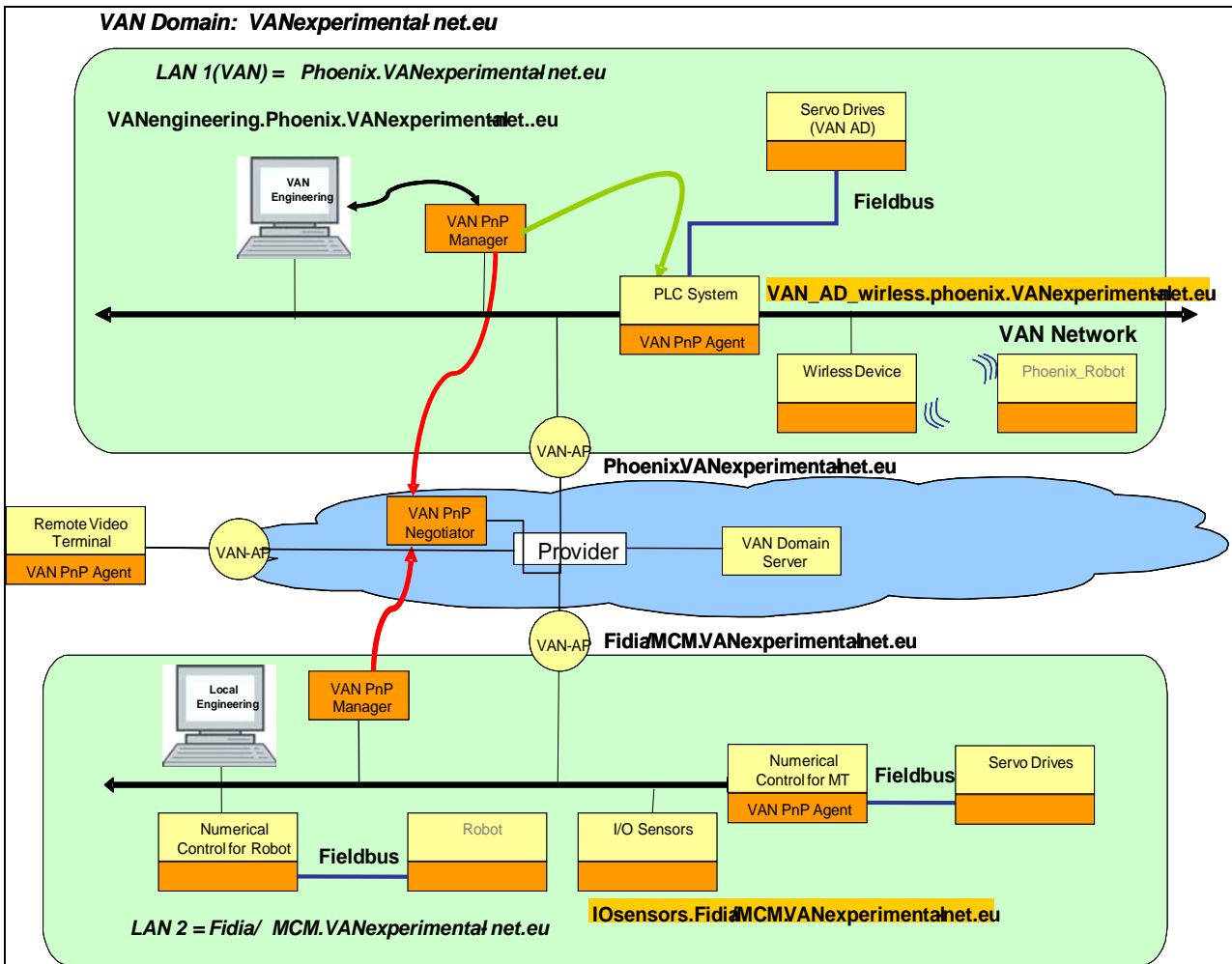


Figure 9: Example of a Factory Automation Topology

To reduce the complexity of this example only components that are needed for work package WP8 are picked up. Figure 10 shows a reduced example, which can be engineered with a VAN Engineering Tool based on the VAN Information Model. A further step as part of the modelling and planning phase is to create a VAN Instance Model. This approach corresponds with steps in the engineering workflow introduced in [D08.2-1].

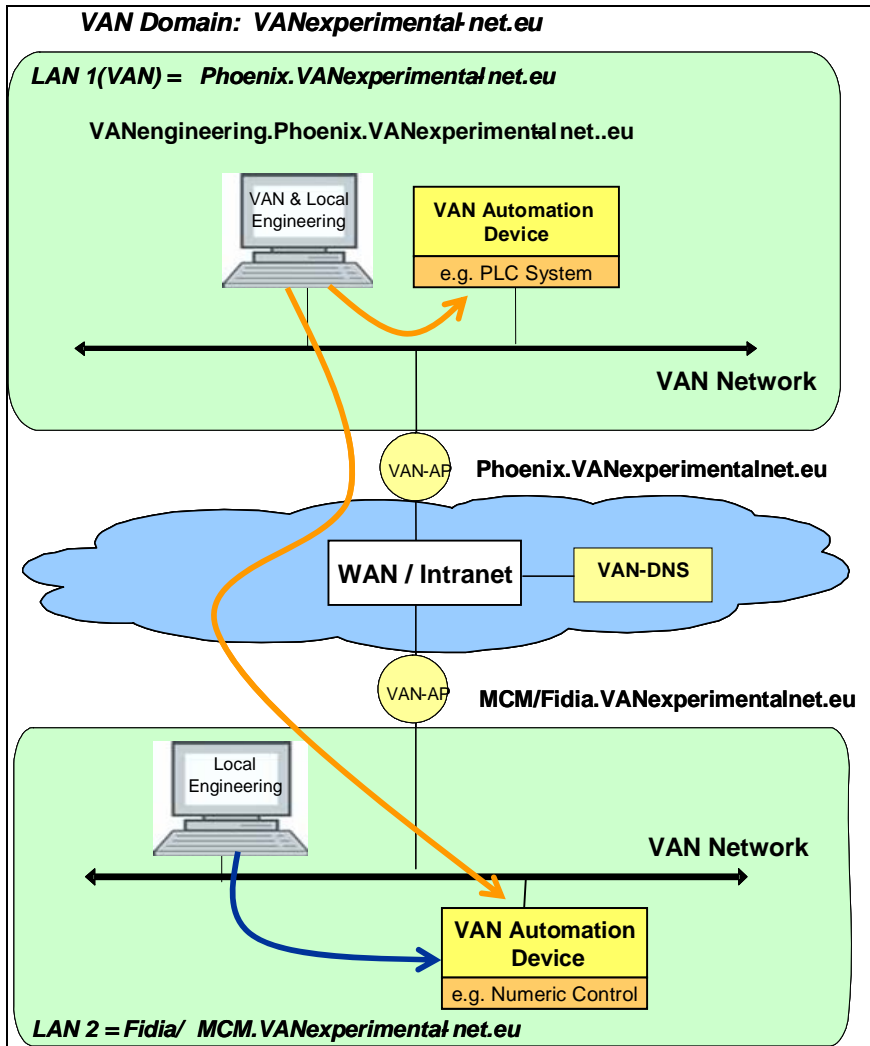


Figure 10: Reduced Example of a Factory Automation Topology

With the use case shown in Figure 10 a testing area to proof the concepts of VAN engineering and for testing the prototypes described in the following section is available. This test scenario allows the configuration of a VAN device directly connected to the same network and also the configuration of VAN devices via a wide area network.

## 2.4 Realization of a VAN Engineering Tool Prototype

### 2.4.1 Prototype for Stand-Alone Concept

In chapter 2.2.1, the concept of a stand-alone solution is described for supporting the engineering process of a VAN system. The description highlights that it is possible to develop a vendor independent tool, which is able, based on its modular design to fulfil the defined VAN-ECS role as well as the communication role of a VAN-ECD.

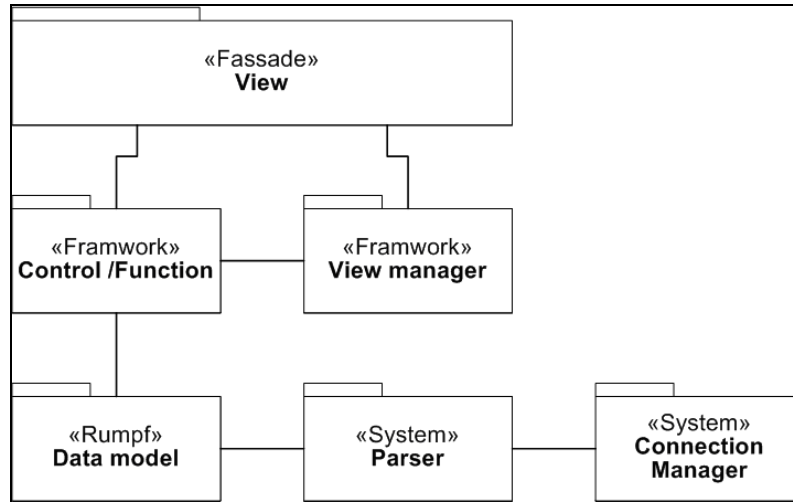
The next paragraph explains the approach to design and code a stand-alone prototype.

Because of its different functions, the first step is the careful design of the tool. Of top most importance here is a design approach that incorporates the possibility for extensions. Mainly, three reasons necessitate this approach. First, if the implementation step will be extended to cover also functionalities of a VAN-ECS, it has to be easy to integrate further function. Second, if the tool is easily extendable, then it is also easy to reuse its components. Finally, in the future other functions may be added. This design approach is primary based on the previously mentioned modular design. Careful documenting and well thought out components are also necessary.

Another important design step is the user interface that will be often used. Therefore, every needless added interaction step is much multiplied. This increases the time to work with the corresponding tool and adds to the cost. In addition, every error in using the interfaces and every too long or too complicated interaction move (for instance successive mouse clicks at opposite corners of the screen) lowers the efficiency of the User Interface. Another important factor in the "easy to work with" capability is the easy of learning.

A well-structured stand-alone solution for a GUI tool should be based on the MVC paradigm. MVC means Model View Controller. The view part implements the User Interface. Primary aspect is here the relief of the user. The control part includes all mechanisms for converting input into action and reverse. This could be for instance the handling of user input and the administration of external tools. The model contains the handled structure. In this context, it may be the data of connections and devices.

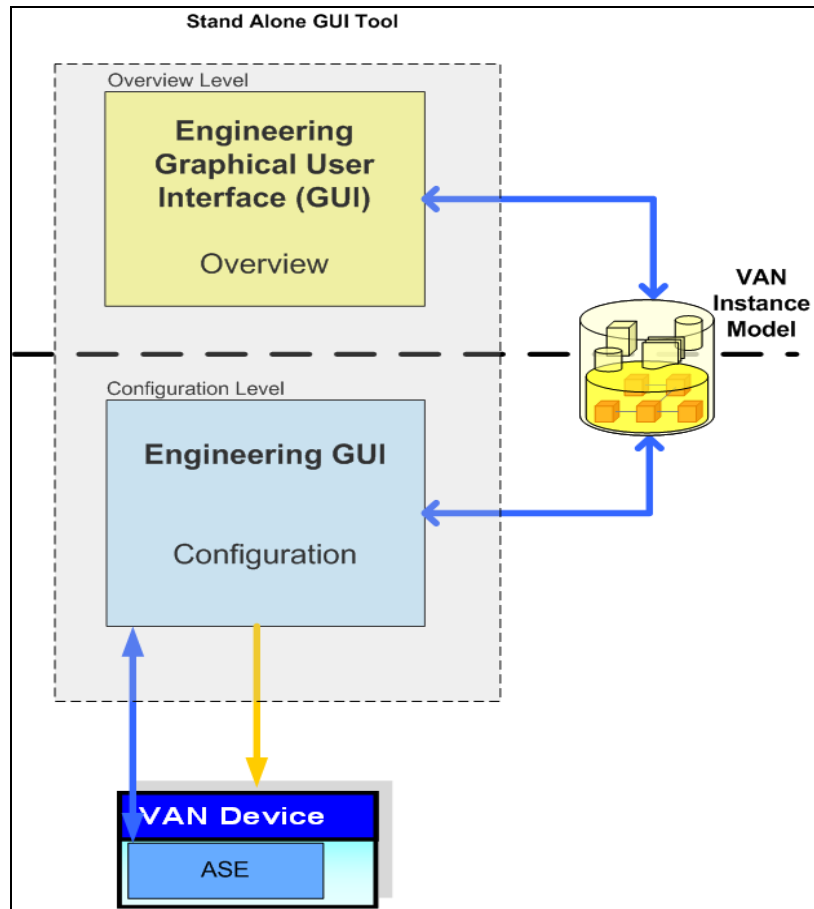
In using components for every main function assemblage like the MVC model modules it is possible to reuse these components. Therefore, it could be possible to show other data structures with the user interface, if it could put into the data model of the MVC. In a first outline (as shown in Figure 11 ) three other components with an importance are defined. The first one is a view manager to handle the different possible views of the editor, for instance the dynamic parts of every ASE. Later on, it is planned to cover different level functions. The second one is the so-called parser. A parser is a program part that converts the received web service data to “in program” data model data. For the prototyping phase, the third program part (the connection manager) manages the connections to different data sources.



**Figure 11: First Outline of Tool Components**

For designing and coding such a prototype with its described functionality, every component has to be set out in writing full and exactly documented. In addition, the definition of the interfaces between components and classes must be specified. For reuse of these interfaces, a well documentation is also important.

The Figure 12 shows a schema of a possible stand-alone solution with its interfaces to fulfil the described tasks.



**Figure 12: Schema of a Stand-Alone Engineering Solution**

The interfaces are:

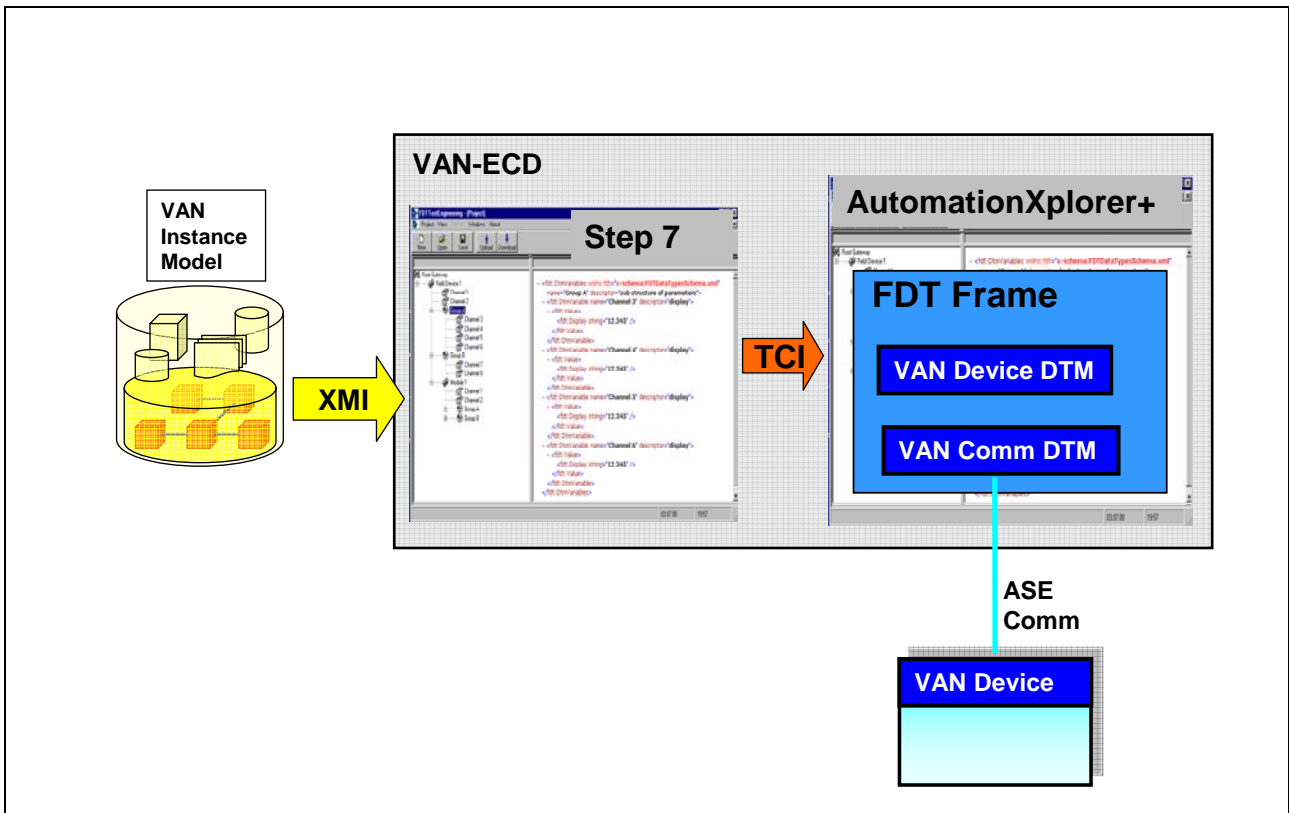
- Interface to web service for parameterization
- Interface to VAN device
- Interface to specified VAN Instance Model

The development and realization of this prototype will be done by means of JAVA and Eclipse [ECLIPSE]. The decision for using this language and tool are following. First, both, programming language and tool, are platform independent. That means they are usable with the most widely spread used operating systems (like WINDOWS, LINUX and MacOS). Furthermore, the developing system is also available free of charge.

JAVA is a programming language, that can relatively easy integrate or be integrated with other languages like C(++). This is possible by means of interfaces like JNI (Java Native Interface). The Eclipse tool is comprised of all necessary parts for development. These tools are CVS (a tool for version controlling of program source), GEF (the Graphical Editing Framework) and other useful functions for programming. This all will make the reuse and further use after the first implementation steps much easier.

### 2.4.2 Prototype for Integrated Concept

Chapter 2.2 explains the main idea for the realization of a VAN-ECD. Figure 13 shows on the left hand side the VAN Instance Model that can be regarded as the original source of information. As a prototype has the main purpose to show the feasibility of the concept each UML tool being able to export its model as XMI code can play the role of the VAN-ECS building the VAN Instance Model. Afterwards a VAN-ECD has to import the XMI file. For the prototype two tools together play the role of the VAN-ECD, namely Step7 of Siemens and the AutomationXplorer+ of Phoenix Contact. It will be shown that a combination of Step7 and AutomationXplorer+ as VAN-ECD already meets many requirements so that it makes sense to extend these tools with the features needed to act as engineering tools for VAN segments. However, it should be noted that other engineering tools might replace Step7 and the AutomationXplorer+ as standard interfaces are used for information interchange.



**Figure 13: Configuration of VAN Devices Using TCI and FDT Frame Application**

Step7 imports the VAN Instance Model using a XMI file. As a XMI file meets the XML standard this interface is a standard interface so that different tools can be used to generate the VAN Instance Model.

The configuration of the VAN devices is not executed directly by Step7; the configuration task is delegated to the AutomationXplorer+ that is called using the Tool Calling Interface (for a detailed explanation see chapter 4). That is, AutomationXplorer+ is entered in the registry as Device Tool for each VAN device. The information is passed in a Temporary Parameter File (TPF). Additionally the AutomationXplorer+ is gained access to the VAN Instance Model. To provide a unique interface for the configuration of the devices the AutomationXplorer+ provides a FDT Frame Application. A FDT Frame Application is a standard interface employed by the Device Type Managers. The Device Type Manager can be compared with a kind of device driver. However, in our case they are applied for the configuration of a device. As both the functions of the FDT Frame Application and the Device Type Manager are fixed once again a standard is used. Thus, the AutomationXplorer+ is replaceable by each configuration tool that meets the TCI standard on the one side and the FDT/DTM standard on the other.

Using Step7 and the AutomationXplorer+, which are discussed more deeply in chapters 2.4.2.1 and 2.4.2.2, has several advantages. Both tools provide full-fledged configuration tools, which support the configuration process in an optimal way. Additionally they can easily be expanded so that they can also deal with VAN devices as they are owned by VAN consortium members. The combination of these two tools is selected, as the TCI is already part of Step7, so that the configuration information can be passed to the AutomationXplorer+. The AutomationXplorer+ is responsible for the communication with the devices using the FDT/DTM approach.

### 2.4.2.1 Introduction to Step7

Main requirements collected in [D08.1-1] can be grouped into several categories. On the one side, the engineering tool must support VAN Network Segments. That is the VAN-ECD must provide features for the configuration of both the devices and their connections. On the other side, the intended workflow must be supported. That is the engineering tool must provide an optimal access to the data, guarantee consistent data during the complete workflow, and facilitate the engineering of a VAN domain or segment by providing the user with helpful information.

In this section, it will be shown that a combination of Step7 and AutomationXplorer+ as VAN-ECD already meets a lot of the requirements so that it makes sense to extend these tools with the features needed to act as engineering tools for VAN Network Segments. However, it should be noted that other engineering tools might replace Step7 and the AutomationXplorer+ as standard interfaces are used for information interchange. These interfaces are XMI files for passing the VAN Instance Model from the VAN-ECS to the VAN-ECD, TCI for sending the information from Step7 to the AutomationXplorer+, and the FDT/DTM for the configuration of the VAN devices.

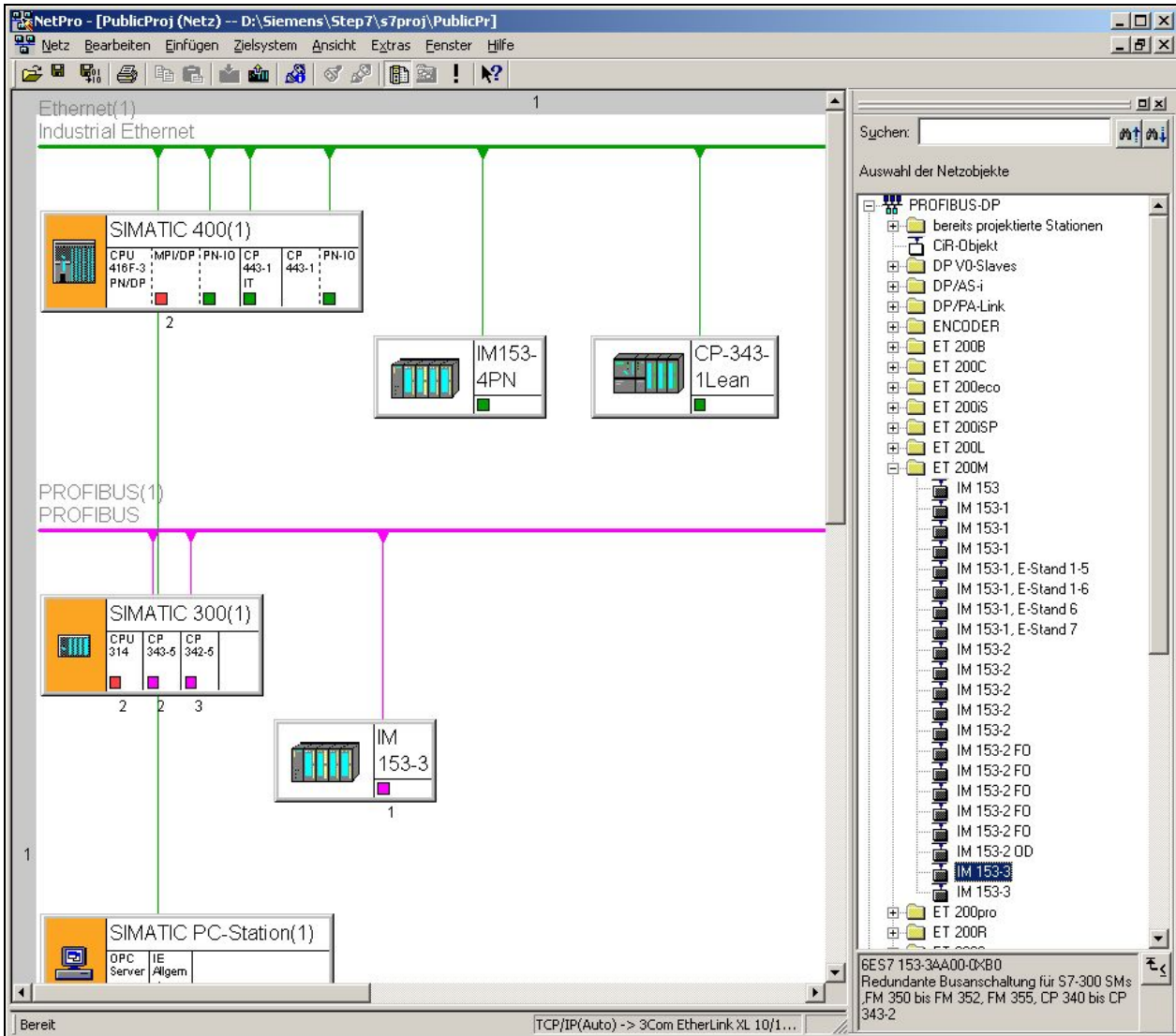
Currently Step7 offers many different features to the user. On the one hand, an interface is provided to the user, which shows all the devices and connections. Thus, Step7 provides a good overview of the network and all of the devices being part of the VAN segment. The available devices are listed on the right hand side of the user interface (compare Figure 14), new devices are added via drag and drop to the network. Part of the engineered network is also a module containing a CPU, i.e. Step7 makes also programming capability available. The programming can be done in three different programming languages, e.g. in ladder logic, using a statement list, or with function block diagrams. For a detailed introduction see [Step7Int] that is part of the documentation delivered with Step7.

It is intended to set up a new import interface for UML models for Step7 using the XML Metadata Interchange (XMI) standard. Thus, any UML tool that is able to export their model as XMI file may generate the VAN Instance Model. An export mechanism to write the changes made in Step7 back to the VAN Instance Model would be useful. However, for the prototype it is concentrated to show the feasibility of our concept. Therefore, it is not intended to include an export mechanism.

Beside the import interface, the VAN devices have to be added to the list of available hardware. As usual, the devices are described by a device description. Nevertheless, in the case of VAN devices a VAN specific device description is used in addition to the GSDML (General Station Description Markup Language) file. Provided that the internals of Step7 are adapted to the new kind of device description Step7 is now able to represent the network structure at the user interface. It is the aim that the connections between VAN-aware and VAN-enabled devices (see the definition in [D02.2-1], page 15) are handled in the same way as conventional networks. It will be possible to represent different types of communication systems, e.g. Profibus, Profinet and VAN-networks at the same time. Thus, Step7 can be used as an engineering tool for established types of networks and devices. Important features, available in Step7, include a list of available devices that might be used during engineering and a user interface, which shows both the configured devices and the connections between them. Additionally the VAN Instance Model has to be placed in the project directory, so that the AutomationXplorer+ has access to it.

In Step7 additional information, e.g. about an already existing network, can be added. Afterwards the VAN specific information will be transferred to AutomationXplorer+ via the Tool Calling Interface. As the information in the TPF depends on the type of connection, i.e. different attributes are passed for Profinet and Profibus, it may be necessary to define an own project-internal standard for VAN devices. To be able to configure real VAN devices it is necessary to extend the TCI so that the set of transferred attributes does not depend solely on the bus category, but on the device type. The

information which attributes are passed in the Temporary Parameter File is taken from the VAN Device Description. For a first approach, a set of attributes common to all VAN devices has to be defined. So first, a list of common attributes for all VAN devices is implemented. To select these attributes the Application Programming Interface (API), specified in the deliverable [D02.2-2] is helpful. Particularly the two methods *SetConfig()* and *GetConfig()*, used for configuration and for monitoring of a configuration, are of special interest, because Step7 must provide the input parameters for them. The most important attribute is a unique identifier of the device. Further attributes are necessary for special types of VAN-devices. But it is not possible to give a complete list of attributes here, as each VAN device may implement different ASE, which requires different attributes.



**Figure 14: User Interface of Step7**

The AutomationXplorer+ as the receiver of the TPF is responsible for the configuration of the devices. For the configuration of the VAN devices web services are used as described in [D02.2-2].

Finally, a short summarize of the changes that are necessary at Step7 to act as part of the VAN-ECD:

- An import interface for the XML files has to be added, so that Step7 is able to read the VAN Instance Model. An export mechanism is useful, but is not implemented in the prototype.
- A new type of device description has to be defined for the VAN devices. To be able to read the device description the functions in Step7 that interpret the device descriptions are extended, so that they can deal also with the VAN Device Description. The TCI must be aware of a set of VAN attributes. Usually the set of attributes does not depend on the device, but only on the type of connection.

- As soon as VAN devices and possible connections are declared, Step7 is prepared for the definition of a domain including VAN-devices.

### 2.4.2.2 Introduction to AutomationXplorer+

The AutomationXplorer+ is a stand-alone FDT Frame Application with extended capabilities concerning TCI. So DTMs can be used for configuration, parameterization or diagnosis of devices from a TCI based engineering system like Step7.

The FDT/DTM specification describes a component-oriented technology to setup and run networks and devices. It describes the interfaces between the components independent of any field bus protocols. The annexes contain the enhancements concerning to specific field bus protocols. Therefore, there can be added new types of networks by creating a new annex to the FDT/DTM specification.

One component within the FDT project represents one device in the real world. A Device DTM represents an intelligent device, which can be parameterized. It is used to setup, configure or parameterize the device. A Communication DTM or a Gateway DTM represents an intelligent device that provides a link between two different networks. FDT uses an abstracted mechanism and well-defined mechanism to communicate with the devices. The Communication or Gateway DTM realizes the corresponding link on the software side between the networks.

FDT provides with the capability of nested communication a solution not only to communicate with devices that are attached to the same network as the PC running the engineering system but also to communicate with devices that are indirectly attached to that network.

The runtime environment for the software components (DTMs) is a FDT Frame Application. It provides a user interface for the network topology realizes the project management and hosts the DTMs and their user interfaces.

On the one hand, the AutomationXplorer+ is a plain FDT Frame Application. It provides a runtime environment for DTMs, whereas each DTM is responsible to configure and parameterize exactly one device, VAN in this case. The AutomationXplorer+ provides in the first instance only a project management and an environment to show the specific user interfaces of the DTMs. The described part is completely independent of the used device or network.

On the other side, the AutomationXplorer+ realizes a mechanism that allows the FDT Frame Application to be used as Device Tools via the TCI interface. Therefore, the AutomationXplorer+ acts as a mediator between TCI and FDT. The mediator summarizes several aspects:

- The provision of DTMs, hosted by the AutomationXplorer+, as Device Tool for TCI based engineering systems. Each DTM provides information within its DTM Information document about the devices it works with. The AutomationXplorer+ consumes that information in the DTM catalogue as well as it creates entries within the Windows Registry for usage with TCI.
- Interpretation of the TPF file. The TCI based engineering system creates a TPF file with necessary information concerning to the selected device as well as all other devices within the network. Each file entry contains the device type information and several device instance data. The instance data summarize unique identifier, device address and device parameters (key values pairs). The AutomationXplorer+ interprets the TPF file in that way, that it creates new DTM instances using the given device types for each device instance that is not known within the AutomationXplorer+ project. The device instance identifier from the TPF file is reused as FDT system tag. Further information such as device addresses and parameters will be passed to the corresponding DTM instances.
- The AutomationXplorer+ is capable of multiple device configurations. That means, all devices from the Step7 project, whose are attached to the same network and come with a DTM, appear in the TPF file and in the AutomationXplorer+ project.

Possibilities and requirements for VAN specific enhancements result in the following points:

- Modelling and mapping of VAN devices to corresponding Device DTMs

- VAN specific address information within the TPF file and provision to the DTMs
- VAN specific device parameters (key value pairs) within the TPF file and provision to the DTMs
- Provision of a Communication DTM or a corresponding wrapper for the communication server

Some bus specific information, e.g. device addresses, require a separate handling, because the device address is located at different places within the DTM parameter document for different bus protocols. While Profibus is directly supported, other bus protocols use their own schema and put their addresses within the DTM parameter document below the element *<UserDefinedBus>*.

Further information is simply copied by the AutomationXplorer+ from the TPF file into the DTMs. Each DeviceItem in the TPF file contains a parameter list. On the other side, the DTM should export variables in its parameter document. The naming of the exported variables should be consistent to the naming of the parameters in the TPF file. Therefore, the AutomationXplorer+ can copy the parameter values to the exported variables with the same name.

The AutomationXplorer+ provides its project data as file in a directory as specified by the TPF file. The directory may be located on a local or a network drive.

In principle, different types for storage are possible. The FDT Frame Application has no special requirements referring to this.

## 2.5 Common Data

Chapter 2.4 describes how Step7 and AutomationXplorer+ act together by using the TCI. Thus, the information between these two tools is passed in a temporary parameter file. For a more detailed explanation of the TCI and the information passing in the TPF, see chapter 4.

The aim of this chapter is to explain which mandatory parameters have to be sent from Step7 to the AutomationXplorer+. Hereby it is necessary to meet the required attributes of the TPF, which are described in Figure 15. Further necessary attributes are contributed by the API definition of the ASE that is used for the configuration of the devices.

According to the definition of the TPF, several device dependent attributes are needed to identify the device and to enable the communication between the device and the tool used for its configuration. These attributes are

- Device Reference. This attribute is part of the XML-element "DeviceItem" and is used for the unique identification of the device. The meaning of this parameter is similar to the Object-reference used in the ASEs.
- Bus Category for the selection of the protocol used for the communication with the device. This attribute is part of the XML-element "General".
- SubsystemID A TPF contains only devices of the same subsystem. This attribute is also part of the element "General"
- SubsystemName Name of the subsystem of the devices in the TPF.
- IsSelected. The data of multiple devices may be transferred in one TPF provided that the same Device Tool is responsible for their configuration. The flag IsSelected marks exactly one device that is selected by the user when the Device Tool is called. It is up to the Device Tool how this flag is handled.

A comparison with Figure 15 shows that this list of mandatory attributes is not complete. Only mandatory attributes depending on the device are mentioned. As an additional parameter the

- ProjectRelatedPath is required. The "ProjectRelatedPath" contains information about a directory, which is assigned to the project context of the Engineering system. This directory may be used to store the VAN Instance Model.

The remaining attributes are explained in chapter 4.3 that deals with the Temporary Parameter File.

The type of communication is described in the conformance class. Conformance class "C1" means that only the call of the Device Tool is initiated, "C2" means that information is passed in the Temporary Parameter File (TPF) and if both Engineering and Device Tool have conformance class "C3" the Engineering system provides a communication server which simplifies the communication between the device and the Device Tool. For the VAN engineering prototype AutomationXplorer+ should act as Device Tool and the parameters are passed from Step7 to AutomationXplorer+ using the TPF.

The information enclosed in the TPF must be sufficient to describe the VAN device to be configured. According to the Device view, presented in [D08.3-1] in chapter 2.3.5, the VAN devices are organized in a hierarchical way. The approach in this section is to select attributes common to all VAN devices and additional attributes, which are necessary to organize the information exchange between Step7 and AutomationXplorer+. The latter list of attributes includes for example the attributes mandatory for TCI. For special VAN devices additional device-specific attributes have to be added depending on the type of the device. It is intended to extend the TCI, particularly the specification of the TPF, so that arbitrary attribute value pairs are passed in the element *ParameterItem* (compare the structure of the TPF file described in Figure 15). Please note that the syntax of the TPF is not changed by this approach. In the current implementation only a bus specific set of attributes is passed. This approach is not sufficient if VAN devices should be configured in Step7.

According to [D02.2-2], page 27, “a VAN device contains a set of object-oriented ASE. Each ASE specification is composed of two parts, its class specification, and its service specification”. Usually the ASE provides the two service methods get and set. “There shall not be additional services beside these get and set services” ([D02.2-2], page 35), so we concentrate on the class definition here.

For the class description of each ASE the object reference has to be given (compare [D02.2-2], page 38). This parameter gives a unique address of the device to be configured. Additionally the type of the device is given by the attribute object type. The third mandatory attribute is called member in groups and is important if a VAN device is able to store multiple parameter sets at the same time. Only one of these parameter sets, determined by the attribute active-group of the class device description, is the active one.

The class *Common Device Config* of the VAN Device Config ASE provides additional attributes that are of large importance for the configuration (see [D02.2-2], page 43). The following attributes are used for its description:

- List-of-device-config-objects (mandatory): This attribute gives a list of neighbouring objects that are identified by their object reference and object type. Both of these attributes are mandatory.
- Manufacturer-identifier (optional)
- Class-version (mandatory): Version of the class specification shall be set to “1.0”. It is expected that this attribute be changed when firmware is updated. Therefore, it makes no sense to pass this attribute in the TPF.
- Configuration version (mandatory): This attribute is incremented when the configuration is changed and set to zero if the factory configuration is used. But in [D02.2-2] it is defined as read only. It is therefore not passed between Step7 and the AutomationXplorer+.
- Interface-version (mandatory): Provides the version of the class specification, shall be set to “1.0”. It is expected that this attribute be changed when firmware is updated. Therefore, it makes no sense to pass this attribute in the TPF.
- Factory reset (optional): If set to true a reset is executed.

When *SetConfig()* is called the only mandatory parameter is the object-reference, which is used for the unique identification of the device. The remaining parameters are passed in a XML-file, so that the information passing mechanism is flexible.

Up to now, we identified attributes that are mandatory to meet the requirements of the TCI and a second group of attributes that are required for the configuration of the device whose data are passed via TCI. An overview of a minimal attribute set is given in Table 1. Additional attributes for a real device are necessary as a real VAN device may provide more specific ASE as described in the general VAN Device Config ASE. Both the attributes required by the TCI and the ASE have to be passed to the AutomationXplorer+. To pass the information required by the structure of the TPF is no problem.

**Table 1: Minimal Set of Attributes passed in the Temporary Parameter File**

<b>Name of the attribute</b>	<b>Type</b>	<b>Remarks</b>	<b>Required By</b>
Object-type	string	A unique identifier for the device type. According to [D02.2-2], page 43, this attribute is optional for the class VAN ASE, but is mandatory when used as part of the attribute "list-of-device-config-objects". So it is recommended to regard the object type as a mandatory attribute.	ASE
Object-reference (Name used in [D02.2-2], page 38) device reference (Name used within the TCI specification)	string	Unique identifier for the device. This attribute is necessary according to the VAN Device Config ASE, [D02.2-2], page 43.  According to the TCI specification, "the attribute "DeviceReference" is used to address a device instance unambiguously and is needed for addressing the communication interface. The Engineering System shall ensure the unique nature of this attribute. The structure of the attribute is only defined by the Engineering system. It is not allowed to interpret the syntax of this keyword in the Device Tool. The Engineering System shall ensure that the content of this keyword does not change, object properties are changed in the Engineering system.	ASE, TCI
Member-in-groups	String	Used to send several parameter sets to the same device. For a complete description of the concepts of multiple parameter sets see [D02.2-2], page 37	ASE
Manufacturer identifier	String	Part of the Common Device Config.	Common Device Config
List-of-device-config-objects	String	This attribute shall contain all neighbouring objects and consists of the attributes object-reference and object-type	Common Device Config
Factory-reset	Boolean	If set true the active configuration will be deleted and the default factory configuration is loaded.	Common Device Config
Active-Group	Unsigned8	This attribute identifies the active set of configuration objects. A write access to this attribute results as a general rule in a change of the current configuration (confer [D02.2-2], page 47)	Class Device Description (confer [D02.2-2], page,47)
BusCategory	String	This attribute is usually used to	TCI

Name of the attribute	Type	Remarks	Required By
		distinguish between Profinet and Profibus. For the intended prototype this attribute can be used to mark a device as a VAN device	
IsSelected	Boolean	Several devices which may be configured by the same Device Tool may be listed in the TPF, but only one of the devices is selected when the Device Tool is called	TCI
ProjectRelatedPath	String	<p>The attribute "ProjectRelatedPath" contains information about a directory which is assigned to the project context of the Engineering System. A Device Tool should use this path for storage of its device data. The format and structure of this data is defined by the Device Tool itself.</p> <p>This directory could be used as a place for the VAN Instance Model</p>	TCI
SubsystemID	String	Unique identifier for the subsystem of the devices in TPF.	TCI
SubsystemName	String	Name of the subsystem of the devices in the TPF.	TCI

The attributes required for the ASE has to be passed as attribute value pairs in the ParameterItem of the TPF. Both the attributes “Name” and “Value” are of type string. So the “Name” could be used to store the attribute’s name and the “Value” to store the intended setting. To use these attributes to pass the intended information the TCI is extended, so that attributes declared in the device description, either the GSDML or the VAN specific device description, are passed. Additionally the VAN Instance Model is placed in a directory, which is passed by the attribute “ProjectRelatedPath” (the “ProjectRelatedPath” is a mandatory attribute of the XML-element “General”, see Figure 15).

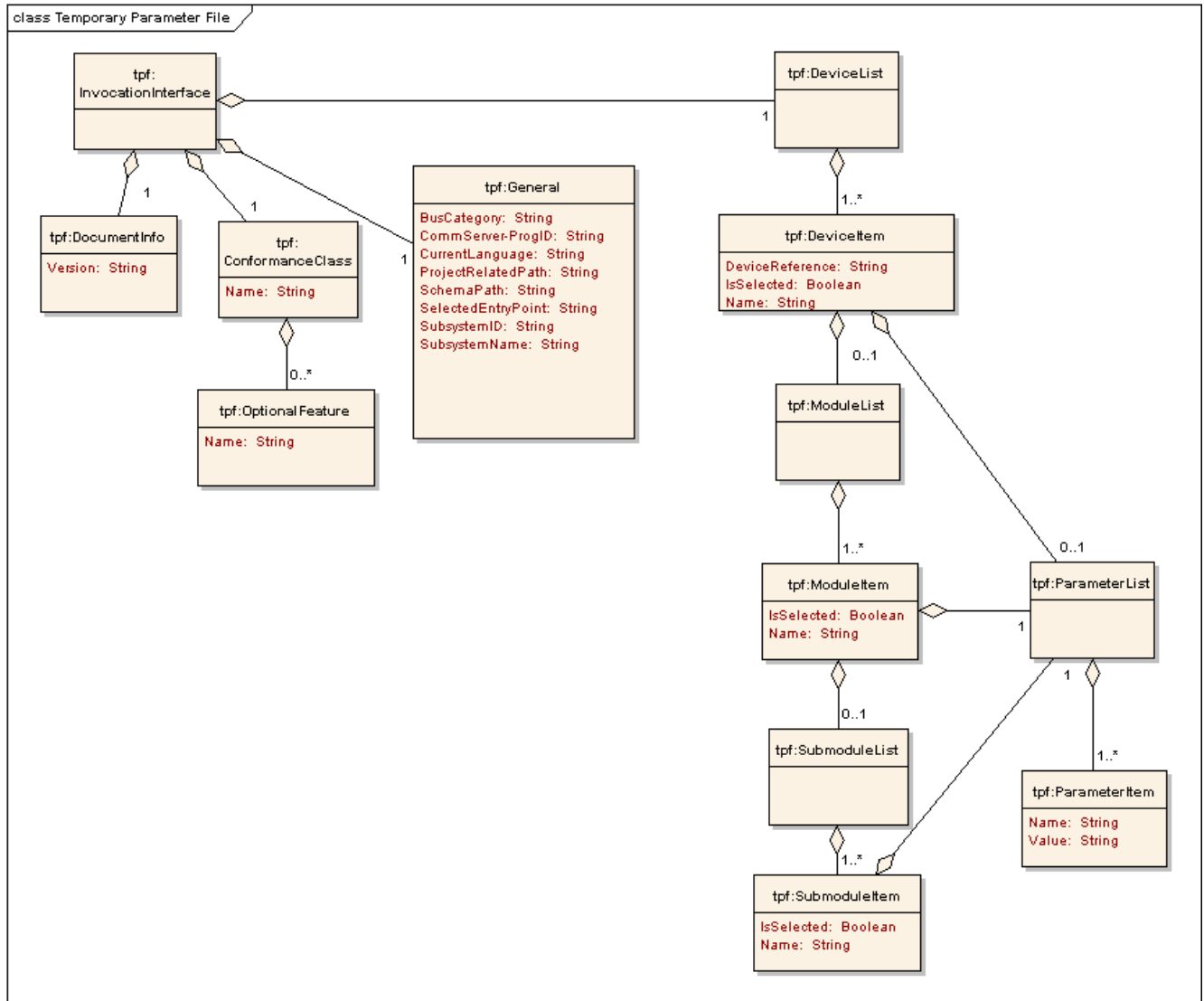


Figure 15: Structure of the Temporary Parameter File

## 3 VAN Device Description (VAN-DD)

### 3.1 Introduction

A Device Description (DD) enables the integration of real product details into the tools of the engineering life cycle. Normally a DD is a generic language for describing the properties of automation system components and covers following features:

- device parameters and their dependencies
- device functions, for example, simulation mode, calibration
- interactions with control devices
- persistent data store.

Device descriptions are used with appropriate tools to generate interpretative code to support parameter handling, operation, and monitoring of automation system components such as remote I/Os, controllers, sensors, and programmable controllers, see Figure 16. A DD specifies the lexical structure, syntax and semantic. The device description file provides a set of supported parameters for a concrete device. A concrete tool implementation is not in the scope of a device description.

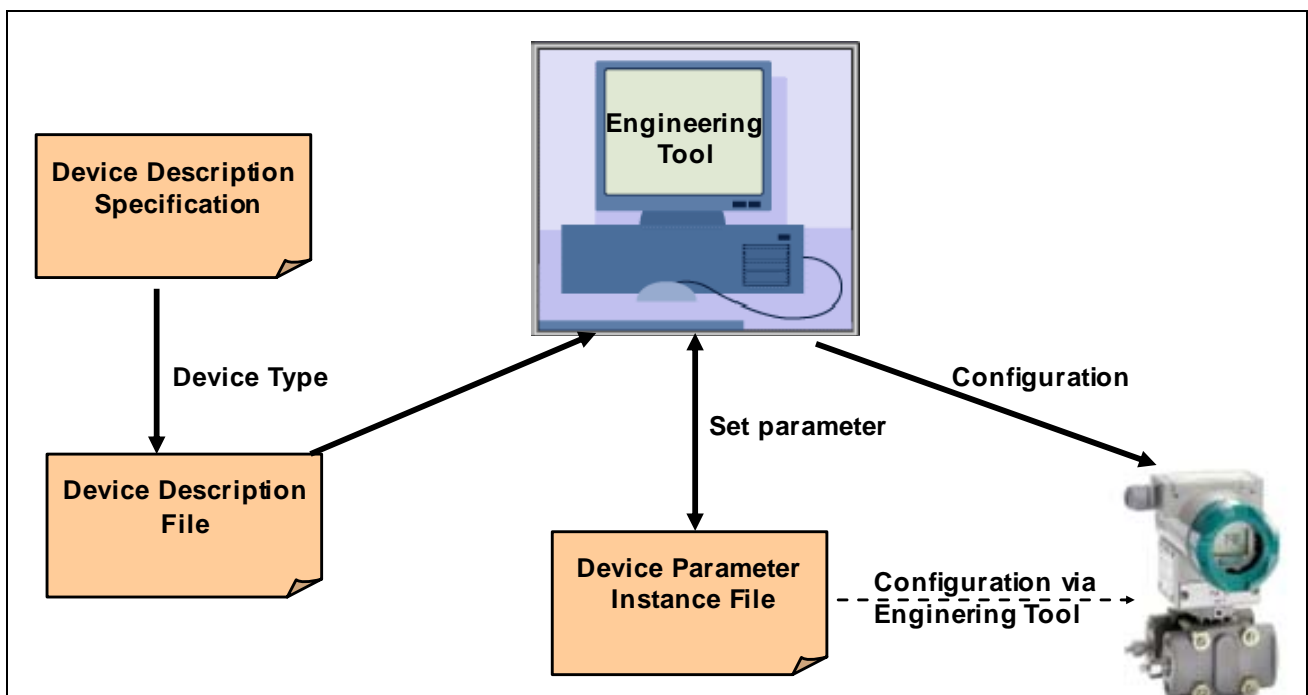


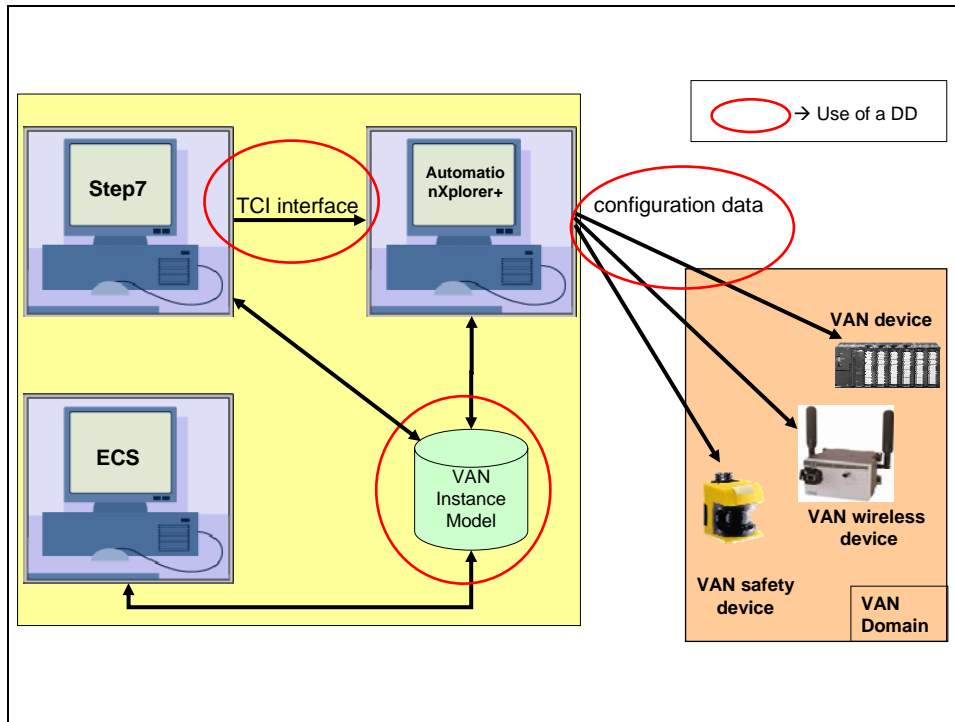
Figure 16: Correlation of a DD to the Engineering Process

For the utilisation of a device description different technologies were analysed in task T8.3, see also [D08.3-1]. The analysis includes GSDML as used description for Profinet, as well as the established XML based solutions DDXML, FDCML. The result on the analysis is to use the parameter of the GSDML, which are provided for every Profinet device by the vendor, and to develop a VAN specific DD solution to describe the data not covered by the GSDML.

### 3.2 Use of a Device Description within the Integrated and Stand-Alone Concept

Concerning the VAN project and especially for the engineering aspects a device description specification for the support of the VAN specific parameter is needed.

For the integrated prototype, see chapter 2.4.2, these specified parameters are used for the description of the configuration parameter in every VAN device, the interaction between the VAN-ECS and the VAN-ECDs and as well for storing the instance information of the devices, see also Figure 17.



**Figure 17: Use of a DD for the Integrated Prototype**

Within this concept only the DD specification is needed for the definition of ASE objects, attributes and their semantics. The derivation of ASE objects and attributes, comparable with the generation of the device description file in Figure 16, will be realized by the VAN-ECS and stored in the VAN Instance Model. This device description can be modified by the imported information from Step7 or the AutomationXplorer+.

Regarding the stand-alone prototype introduced in chapter 2.4.1 a device description is used for providing device type information, see Figure 18. Therefore, the device description file can be read by the VAN-ECD engineering layer. Within the VAN-ECD the device parameters are set. The generation of instance information can be done by using already existing information, which is provided by the ECS engineering layer and stored in the VAN Instance Model or edited in layer 1 the user interface. The configuration of the devices is realized by web services, the actual configuration sets are stored into the VAN Instance Model.

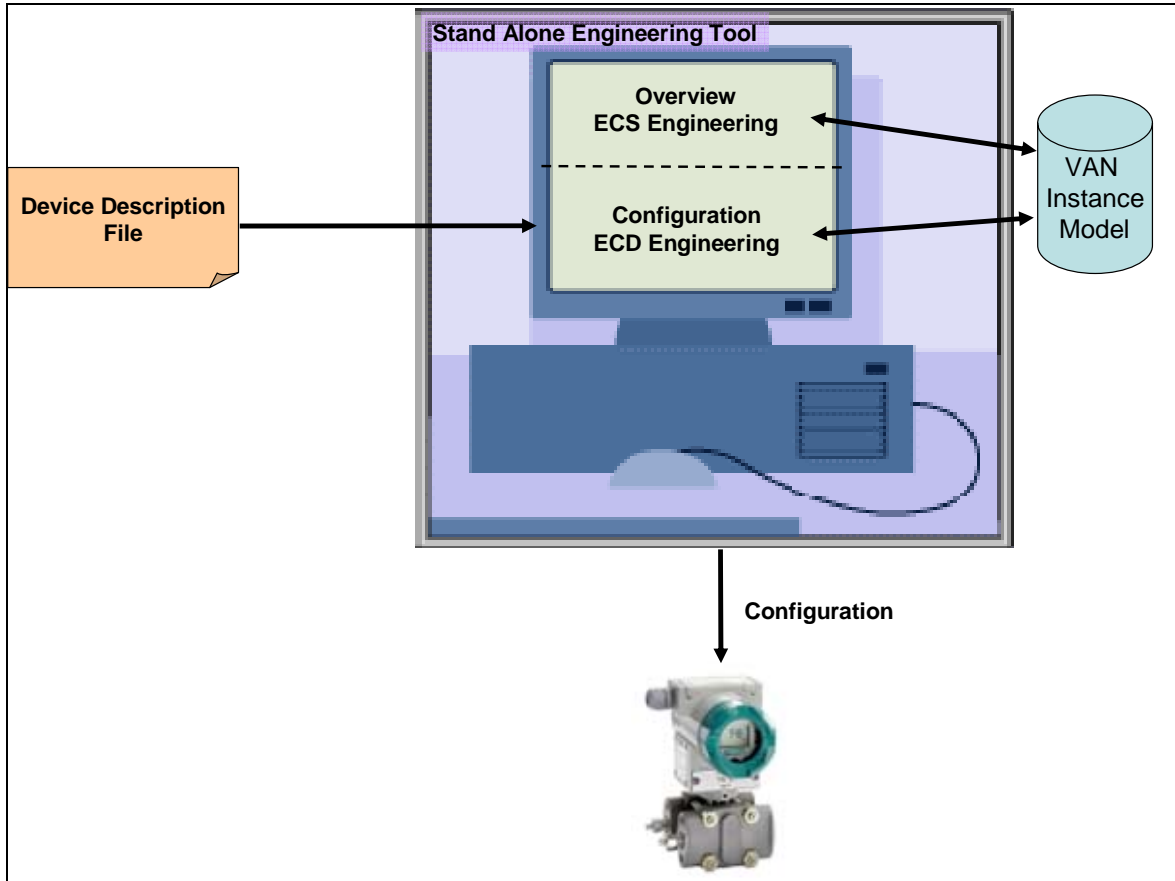


Figure 18: Use of a DD in the Stand-Alone Prototype

### 3.3 Basic Requirements on VAN-DD

As mentioned in the chapter 3.1 the VAN Device Description must be able to represent VAN specific parameters, which are extensions to the Profinet communication. These are basically data of the ASE, which are defined within the work package WP2. Furthermore the mechanisms how to transmit these configuration data from the engineering tool (or VAN-ECD, see Figure 3) into the device has to be specified.

As a result of task T8.3 the following terms have to be met:

- The VAN-DD must be able to describe all features of the VAN Information Model, defined in [D08.3-1]
- The VAN-DD should be compliant to ISO 15745 [ISO15745]
- The VAN-DD should be XML based
- A set of each supported ASE must be included
- The VAN specific parameters have to be downloaded to the device by means of web services

### 3.4 Concept of the VAN-DD

The VAN Device Description provides type information that can be instantiated for configuration and parameterisation of the VAN related parts of a device. The VAN-DD represents an add-on and does not cover application aspects or communication characteristics, which are already defined in the GSDML file and used in the local engineering.

For the VAN-DD, XML will be used due to the following advantages:

- Open standard for description of data
- High acceptance in the IT world
- Unicode based
- Established parsers are free available
- XML schema technology allows a formal description of the document structure
- A large number of XML editing tools are available on the market

ISO 15745 "Industrial automation systems and integration - Open systems application integration framework" defines a set of basic elements and rules for the description of integration models and interoperability profiles. It contains generic information how to build a framework – or in the scope of VAN a device description – and some information for special technologies. Several device descriptions like FDCML, GSDML are based on this standard.

ISO 15745 is applicable for most industrial automation like discrete manufacturing, process automation or electronic assembly.

The general common structure of ISO 15745 is already described in [D08.3-1]. The VAN-DD will be based also on this standard; therefore the entry point and some more general data about the top level structure are predefined. Part 1 "Generic reference description" of this standard includes the basic specifications for a device description. This part should be considered mandatory for the development of the VAN specific device description. Because of the relation of VAN to Profinet communication part 3 "Reference description for IEC 61158 based automation systems" and further part 4 "Reference description for Ethernet based automation systems" should be analysed for the use in VAN.

The VAN-DD contains common information, the definition of supported ASE elements, which are needed for parameterisation of a device, and the service descriptions to transmit instance information from the engineering tool into the device, see Figure 19.

```
VAN-DD-Header
(...)

Description of ASE objects
{
    ASE device_config;
    {
        cardinality;
        list of supported attr
        {
            ...
        }
        service_description: wsd1
    }
    ASE diagnosis
    {
        ...
    }
    ...
}
}
```

**Figure 19: VAN-DD Overview**

In the VAN-DD header the following information will be entered:

- XML schema definition
- Used VAN-DD profile
- Device family (e.g. VAN-AD, VAN-PD)
- Vendor, Author and Version information
- Device identification

The content and structure of the ASE objects is based on deliverable [D02.2-2]. The format of the description is oriented at the definitions for classes in IEC 61158 [IEC61158]. The generic form is shown in Figure 20:

<b>VAN ASE: ASE Name</b>			
CLASS:		Class Name	
CLASS ID:		see (3) below	
PARENT CLASS:		Parent Class Name	
<b>ATTRIBUTES:</b>			
1	(m)	Key Attribute:	Key_Attribute
2	(m)	Attribute:	Attribute_0
3	(m)	Attribute:	Attribute_1
3.1	(m)	Attribute:	Attribute_1_1
3.2	(m)	Attribute:	Attribute_1_2
4	(m)	Attribute:	Attribute_2
4.1	(m)	Attribute:	Attribute_1_2
4.2	...		
5	(m)	Attribute:	Attribute_n
<b>SERVICES:</b>			
1	(m)	OpsService:	Get
2	(m)	OpsService:	Set

**Figure 20: Generic Structure of an ASE**

An ASE can be decomposed into three parts:

- Common ASE definitions
- Supported Attributes
  - Definition of key attributes, attributes, structures and arrays
  - Classification of use: mandatory/ optional/ conditional/ selector
  - Definition of data type
  - Optional: number of enumerated values
- Service definition
  - Two mandatory defined services per ASE: get/set; additional services should be only defined if needed
  - Classification of use: mandatory/ optional/ conditional/ selector

For the implementation of the web services "Web Service Description Language" (WSDL) is used. A web service transmits configuration data between the VAN Engineering Tool and a VAN device. It provides instance information, e.g. the object reference, and the list of supported attributes according to the description of the ASE.

### 3.5 Workflow of Data Generation

For the generation of a complete configuration data set of a device, different activities must be performed as explained in the introduction of this chapter. This section contains a concept for generating the necessary configuration. It is based on the integrated prototype, see 2.4.2, and will be verified by means of the prototype activities of work package WP8. The basic activities are already illustrated in Figure 16 and Figure 17. The following steps have to be performed:

1. Specification of device type information

Primary the set type information for each device must be available. Depending on the device profile, e.g. a VAN automation device or a VAN proxy device, the description provides information about supported ASE objects and parameters and contingently their default values. Regarding other device description technologies, this information is mostly made available by the vendor of the device.

2. Parameterization

Thereafter concrete values have to be assigned for each parameter. This data will be provided:

- from the proprietary engineering tool. In most use cases the local engineer has already designed the automation application and furthermore the hardware configuration of the local VAN domain segment. The transmission between the proprietary engineering tool and the Device Tool will be realized via TCI, see also chapter 4.
- from the VAN-ECS. The VAN engineering provides information about relevant VAN domains and in addition some parameter values must be specified by the VAN engineer.

3. Configuration

The entire parameter set must be transmitted to the VAN device. For the transmission web services are used. Relating to the prototyping of work package WP8 this will be realized by FDT/DTM technology, see also chapter 5.

4. Administration

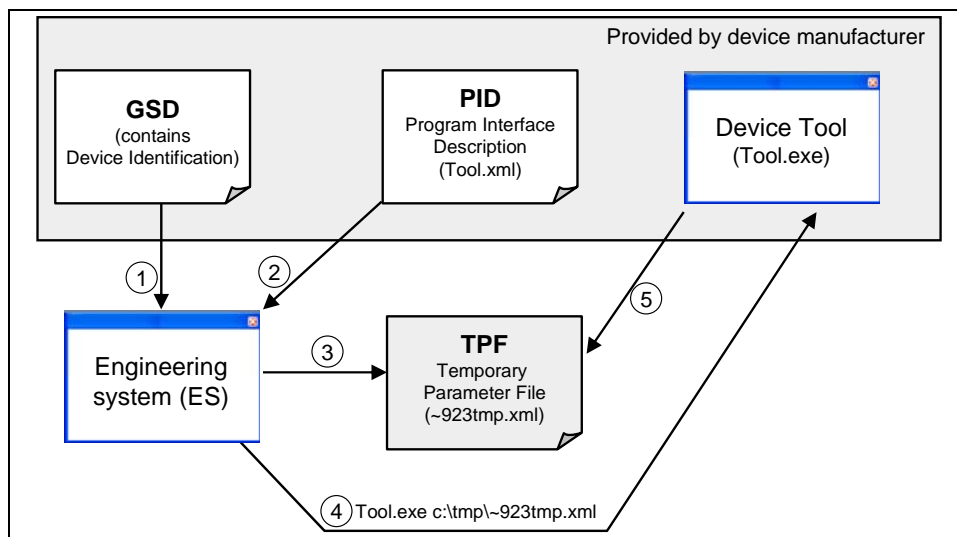
The descriptions of every device must be stored and administrated by the VAN-ECS. For storage a database, file system, etc. can be used.

## 4 Tool Calling Interface (TCI)

### 4.1 Introduction to TCI

The aim of the Tool Calling Interface is to call vendor specific configuration tools directly from the Engineering System (ES).

The TCI approach is also used for the configuration of devices using possibilities not described in the device description. Similar to the FDT/DTM approach vendor specific software, called Device Tools, are used and called directly from the Engineering software. By calling the Device Tool directly from the engineering tool important parameters can be passed from the engineering tool to the Device Tool so that there is no need to enter these parameters twice. The invocation phase and the relationship between the tools and necessary files for the desired information flow are shown in Figure 21 [TCI1].



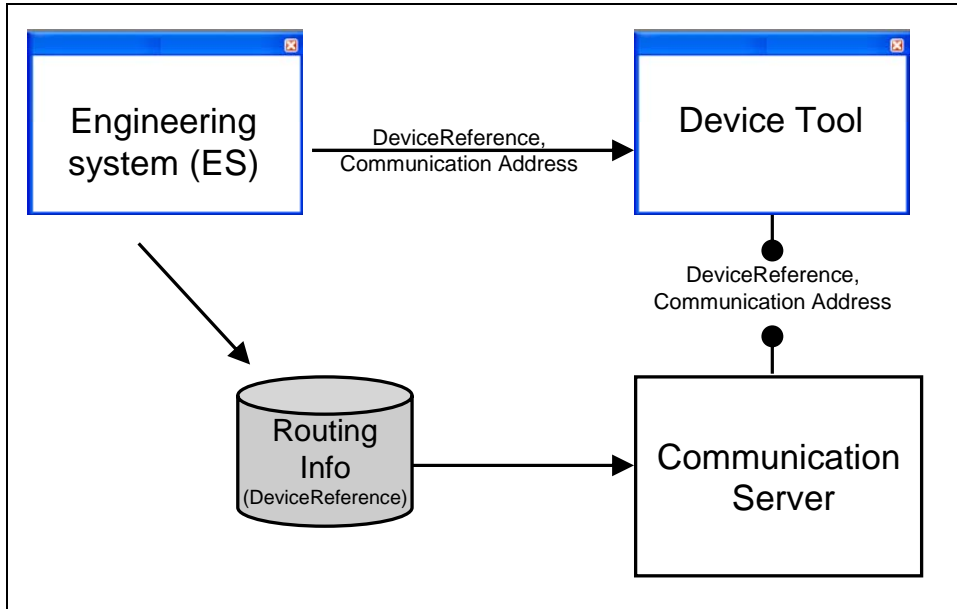
**Figure 21: Invocation Interface – Overview**

1. The GSD file is imported into the engineering system. In the ES, the device is configured and communication settings are made.
2. The engineering system reads the content of the Program Interface Description (PID) file. This file contains information about the interface version and the supported tool functions.
3. Before the Device Tool is started, the engineering system creates a new Temporary Parameter File.
4. The engineering system starts the Device Tool and passes the name of the TPF.
5. The Device Tool interprets the content of the TPF file and deletes the file after processing.

The most common use is the parameterization of a device directly from the engineering system. Such devices will be presented inside the Engineering System by means of GSD or GSDML files. Additionally to the normal modifications of the communication device parameters, application oriented parameter will be covered by external tools. An end user of the engineering system does not expect to invoke a specific Device Tool explicitly. Instead, a special TCI covers such a use case for Profibus and Profinet devices. The invocation of the Device Tool is completely transparent to the user. An additional entry is added to the menu of the engineering system. The menu entries are defined by means of the Program Interface Description (PID) file that is used by the Engineering System. By means of the information of such a file, the Engineering System can prepare the external Device Tool invocation. In order to support the external tool, a Temporary Parameter File (TPF) with device specific

information will be generated and passed as an invocation parameter to the specific Device Tool. It is at the responsibility of the Device Tool to process this file (or not).

When the Device Tool is initialized further customization is possible so that all advantages of the specific tool can be used. If the engineering system provides a communication server (has conformance class "C3") supported also by the Device Tool, also a communication between the Device Tool and the configured device is possible, see Figure 22 [TCI1]. In this case all configuration parameters are directly passed to the device. An additional advantage is that the Device Tool is informed about a directory to store tool specific information. Thus, a general archiving of the information is possible.



**Figure 22: Communication Server Used by the Device Tool**

It is required that the Engineering System and the Device Tool are installed on the same computer. The operating system on this computer must be one of the versions of Microsoft Windows, supporting the Component Object Model functionality. Furthermore the mapping between device types and Device Tools is organized by entries in the registry. For each device type a UUID (Universal Unique Identifier) is entered in the registry. The engineering system uses the UUID to find a description of additional menu entries for the given device and the path to the Device Tool.

## 4.2 Program Interface Description (PID)

As mentioned in chapter 4.1 the PID determines the additional menu entries. Assuming that there is one common menu entry for all VAN devices, only one PID is needed. Please note that this approach does not mean that the same information is passed to the AutomationXplorer+. Both the engineering and the Device Tool distinguish the different VAN devices using the device identifier.

Figure 23 [TC11] describes the structure of the PID. The element DocumentInfo with the Version is not VAN specific. Currently "V1.0" is used.

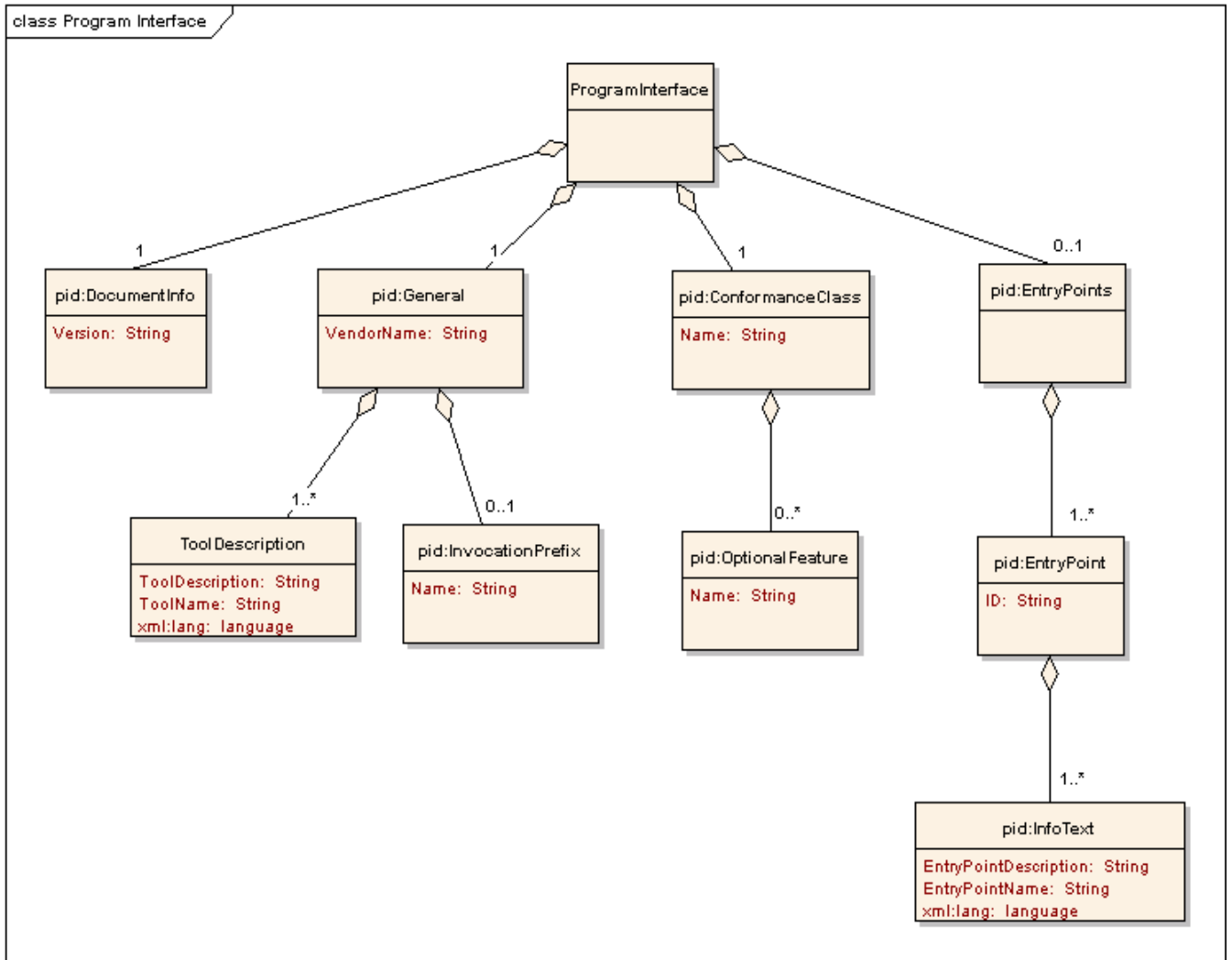


Figure 23: Structure of a PID File

In addition, the element "General" is required. The attribute "VendorName" can be set to "VAN device". As this information is not coded in the TPF, the vendor name can be set arbitrarily.

The element "ToolDescription" defines the menu entries and is therefore visible to the user. The "ToolName" should contain a clear hint that a VAN device is configured, e.g. "Configure VAN device" as "ToolName" for the English language. The invocation prefix must be selected so that the AutomationXplorer+ interprets the following string as path to the TPF.

The conformance class that describes the communication mechanisms is also mandatory. The entry depends on the conformance class of the AutomationXplorer+. The conformance class "C2" is necessary, which means that the Device Tool is able to read the TPF. The conformance class "C3" is possible, but dispensable, as the AutomationXplorer+ is responsible for the communication with the devices.

The element “EntryPoints” is optional. Entry points “are used to generate additional sub menu entries in the “ToolDescription” context menu of the engineering system” [TC11]. As the selected entry point is passed via TPF to the AutomationXplorer+, the entry point can be used to select different functionalities for each device. For example, there could be two functions. The first one checks whether the parameters in Step7 are in conformance with the actual parameters. The second function is used to set the parameters of the device.

The discussed scenario aims only at a prototype to show the feasibility of the engineering process, there is no need to add consistency checks. At the moment, there is no use-case, which justifies this additional effort.

### 4.3 Temporary Parameter File (TPF)

The temporary parameter file is used to pass information from the engineering tool to the Device Tool, i.e. from Step7 to AutomationXplorer+.

Once again, there are some attributes, which are not VAN-specific. The element *DocumentInfo* states the version; currently “V1.0” shall be used.

The conformance class depends on the applied engineering tool. The engineering tool of Siemens, Step7, supports conformance class “C3”. That is a communication server is offered so that the Device Tool has access to the devices.

The element “General” contains some attributes, which are VAN specific. The *BusCategory* specifies the communication protocol between the Device Tool and the device. It provides hints for the Device Tool how the device can be reached. It could also be used to mark a device as a VAN device.

The *CommServerProgID* is optional. As the AutomationXplorer+ takes over the communication with the device, there should be no entry for the *CommServerProgID*.

The *CurrentLanguage* depends on the user environment. E.g., “en” for English or “de” for German is allowed. Other values are defined in ISO 639.

The *ProjectRelatedPath* is used to store data of both the Device Tool and the engineering tool to the same location. This directory can be used for archiving segment related data. This directory should also be used to store the VAN Instance Model so that the AutomationXplorer+ has access to it.

The Schema path is required. It is suggested that the structure of the TPF is not changed. Thus, there is no need to create a VAN specific VAN TPF Schema file. The path to the Schema file can be set arbitrarily.

The selected *EntryPoint* is optional. According to chapter 4.1 it is suggested not to use this feature.

The *SubsystemID* and the *Subsystem* are required, but it seems that both attributes have a Profibus specific meaning. In the VAN context, a unique ID may be used for each VAN-segment. However, this approach might cause problems as a VAN segment to be configured is not necessarily aware of other VAN segments, thus problems might occur in keeping the IDs unique.

The Device List is required and denotes unambiguously one or more devices to be configured. The Device References identify the devices, it is the task of the engineering system to ensure “the unique nature of this attribute”. The flexibility of the TPF approach is expressed by the element *ParameterList* containing at least one *ParameterItem*. A *ParameterItem* is an attribute-value pair where both the attribute’s name and the value are of type string.

## 5 FDT for VAN-Engineering

### 5.1 Introduction to FDT/DTM

(FDT) Field Device Tool/ (DTM) Device Type Manager is an established technology in the factory and process automation area. It is promoted by the FDT Group [FDT Group], a collaboration of international companies, dedicated to establish an international standard for the configuration and parameterization of field devices. The key feature of the FDT/DTM technology is its independence from the communication protocol and the software environment of either the device or the host system. FDT/DTM allows any device to be configured and to be accessed from any host through any protocol [FDT].

The FDT/DTM architecture is composed mainly of two elements: the FDT Frame Application and the DTM. Moreover, the DTMs can be subdivided in Device DTMs, Communication DTMs, and Gateway DTMs.

The FDT Frame Application is a logical object to represent an environment like an engineering system or a stand-alone tool. The FDT Frame Application provides the complete functionality to manage data, to communicate with the device and to embed DTMs. A DTM must be independent of the environment it is running in. So all environment-specific tasks must be handled by the FDT Frame Application.

A DTM may represent different types of devices, e.g. field devices, communication devices and gateway devices.

A Device DTM represents a 'normal' field device. Inside the FDT Frame Application the Device DTMs are instantiated and used part or the complete system. Each Device DTM provides graphical interfaces that permit an easy configuration and monitoring of a specific device. A Device DTM uses a Communication-channel to communicate with the related field device. The Communication-channel physically can be a PC I/O board with processing units and proprietary bus systems.

A Communication DTM represents a communication device that provides communication capabilities via Communication-channels.

A Gateway DTM is a Communication DTM which provide more than one Communication-channel.

In the VAN context the focus is not on the automation devices themselves but on the VAN profiles defined inside the VAN architecture of work package WP2 [D02.3-1]. Each device profile can be represented within a FDT Device DTM. The Communication DTM provides communication functionalities to access the VAN devices using the profile defined ASEs.

Figure 24 shows the representation of a simple automation system from the FDT point of view. The right hand side shows the physical devices, whereas the left hand side shows the representation of the corresponding components in the FDT context.

A VAN Device DTM, which represents the VAN device, is deployed in the FDT Frame Application. As well a VAN Communication DTM (in Figure 24 VAN Comm DTM) is instantiated. The VAN Communication DTM integrates the corresponding VAN Communication-channel. The communication between the VAN Device DTM and the VAN device is established.

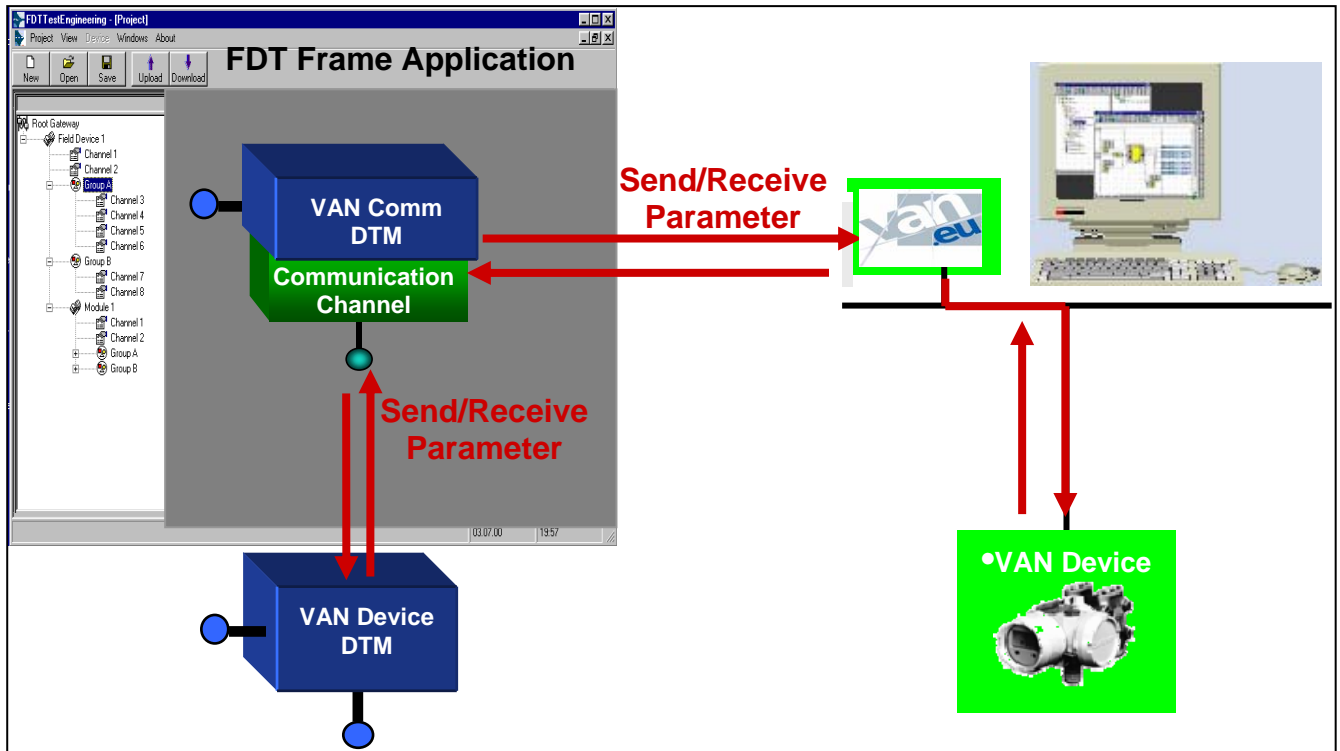


Figure 24: FDT Architecture

## 5.2 Tasks for the FDT Components in VAN

This chapter describes the tasks of the FDT/DTM related components used in the integrated concept with TCI and FDT/DTM as shown in Figure 8. It is also explained how the FDT interface works in general and how these components are used within the VAN context. According to the workflow that is described in chapter 2.1 the VAN domains and the VAN segments of these VAN domains are planned in the VAN-ECS. Afterwards this information is passed to the proprietary tool used as VAN-ECD, i.e. to Step 7 in the VAN Engineering Tool prototype as described in chapter 2.4.2. Within Step7 the setting of additional attributes is performed. After this step is finished, the configuration data is passed to the FDT Frame Application, i.e. to the AutomationXplorer+. This tool has the task to convey the information to the devices by means of the VAN Device DTM and the VAN Communication DTM. The FDT Frame Application and the DTMs have to provide a number of mandatory interfaces that are described in chapter 0.

In the integrated concept with TCI the FDT Frame Application, i.e. the AutomationXplorer+, must receive the TPF and decode the information contained in it. Additional tasks for the FDT Frame Application are:

- As the VAN Instance Model is considered as the main source of information, the FDT/DTM needs also access to the VAN Instance Model. The main purpose of the TCI call is to pass the mandatory parameters described in chapter 2.5 to the Frame Application, so that Frame Application knows which device has to be configured. The current parameter values, not including the identification of the device, are taken from the VAN Instance Model.

- As there is no possibility to send error messages back to the VAN-ECD that is responsible for the invocation of the TCI call, the FDT Frame Application must display error messages directly to the user. This is necessary for example if a DTM is missing.

Usually there are additional scenarios for a FDT Frame Application. These scenarios are discussed in [FDT\_Tec] and therefore they are not VAN specific. Some of the scenarios are listed in Table 2 to give an impression of the features that are required by the user. The use cases mentioned in Table 2 cannot be compared with the use cases for VAN. The FDT Frame Application is a mean for the configuration of the VAN devices. The use cases for VAN describe the usage of the VAN system as a whole. Therefore, real time, safety and security are clearly part of the VAN use cases, but they are not relevant within the FDT/DTM scenarios.

**Table 2: Possible Scenarios of a FDT Frame Application**

Use case	Description
Configuration of devices	Uploading the configuration to the devices using the DTMs. Part of the configuration should be a consistency check, i.e. the attribute settings have to be compatible with the attributes of other devices and with the device description.
Diagnosis	Collecting online data from the device.
Access to VAN segment via proxies	Several Gateway DTMs are required which act as bridges e.g. between multiple segments. These Gateway DTMs might play a similar role within the FDT Frame Application as the Proxy Devices to connect different segments, e.g. an Ethernet or Profibus segment with a VAN segment.
Engineering and comparison of engineered to actual topology	The FDT Frame Application should support planning the topology, scanning the topology to compare the plan with the reality. The configuration of the devices is prepared in offline mode.

After the FDT Frame Application has collected all configuration information about the devices, this configuration information must be encoded in a XML file, which is the standard for information interchange between the FDT Frame Application and DTMs. After the generation of this file, the correct DTM has to be selected, so that the information is passed to the device. Comparing this scenario for the configuration of VAN devices with the use cases described e.g. in [HadTNC] it gets clear that usually a FDT Frame Application has a much broader range of tasks to fulfil. Beside the configuration of devices, a FDT Frame Application must also be able to support the user in diagnosis, scanning the network, and storing the information in a database. Some of these use-cases are listed in Table 2. Diagnosis means that the user is able to request information about the state of a device, e.g. about the actual parameter settings.

The FDT/DTM standard requires the realization of several interfaces from both the FDT Frame Application and the DTMs that are used by the FDT Frame Application. [FDT\_Tec] lists the following interfaces as mandatory:

- The main interface of a FDT Frame Application is the *IFdtContainer*. This interface provides function for the instance data management. Inside a multi user system, it can also be used by a DTM to lock its data set for exclusive write access [FDT\_Tec].

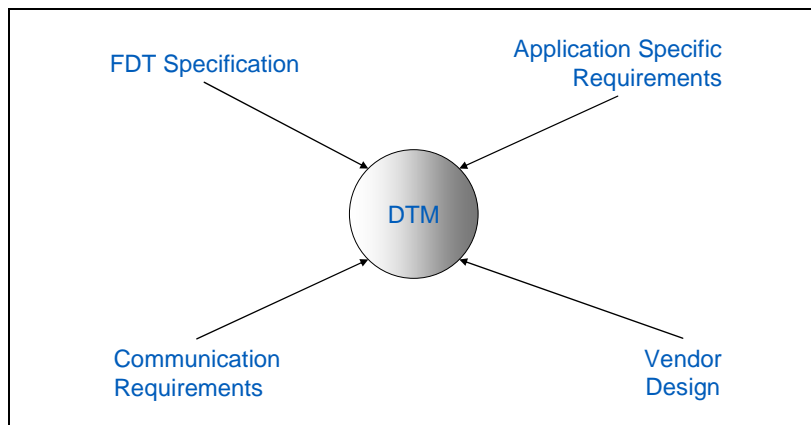
- The *IDtmEvents* of a FDT Frame Application is used by the DTMs to give information about the change of their online state, their available functions or parameters, and the completion of executed functions.
- Via *IFdtDialog*, the DTM may open a dialog to show messages like errors, warnings, or information.
- With *IFdtTopology*, the FDT Frame Application provides access to the system topology. Thus, the Communication DTM can get information about the child DTMs connected to its channels as well as it can add or remove child DTMs from its channels.
- For DTMs providing audit trail information, the FDT Frame Application implements *IDtmAuditTrailEvents*. The FDT Frame Application can then record the historical data, so that the engineering tool keeps track of changes.

Usually these functions are not sufficient for all of the use cases listed in Table 2. Additional optional interfaces are necessary. E.g., there is an interface named *IDtmScanEvents*, which supports the scanning of the segment. As far as the configuration of the devices is concerned the two different interfaces *IDtmSingleDeviceDataAccessEvents* and *IDtmSingleInstanceDataAccessEvents* have to be implemented. It has to be discussed whether a prototype requires full implementation of all mandatory functions, because a subset might be sufficient to demonstrate the concept of VAN Engineering Tool.

### 5.3 VAN Device DTM

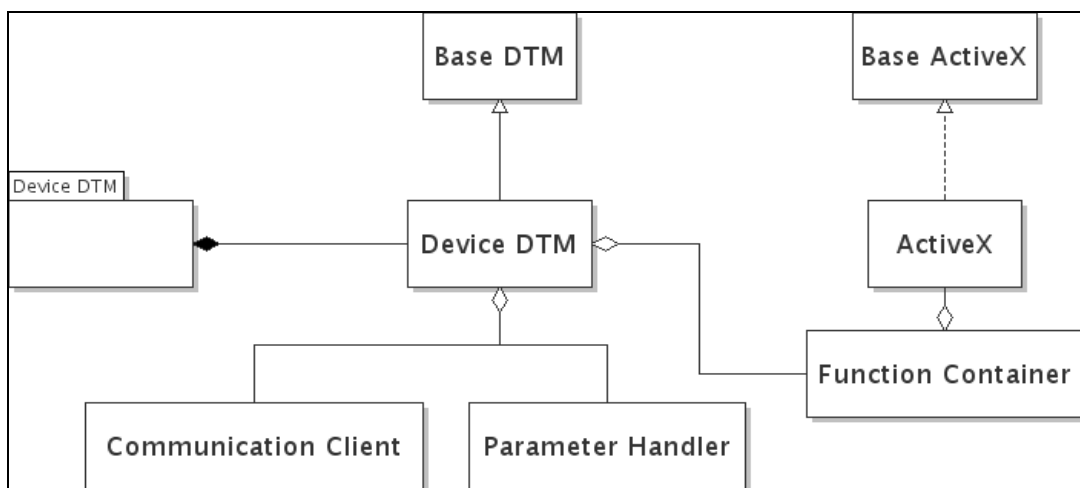
#### 5.3.1 Internal Architecture of a VAN Device DTM

Several aspects and requirements will influence the architecture of a VAN Device DTM (see Figure 25).



**Figure 25: Requirements for a VAN Device DTM**

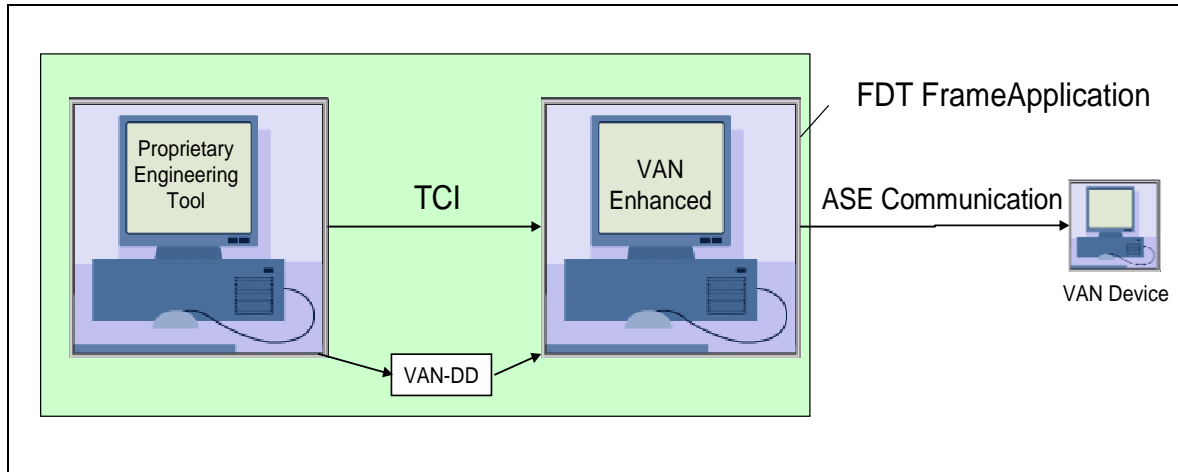
A VAN Device DTM can be developed either from scratch or based on precast libraries. Such libraries e.g. are based on templates of predefined objects. By means of such a development approach, several common interface aspects or communication functions can be reused. Figure 26 shows the internal structure of a VAN Device DTM. All DTM basics are covered in Base DTM. The specific relationships between parameters (hold in Parameter Handler) and different ActiveX controls will be managed by the VAN Device DTM itself. For the communication with the device, predefined functionality depending on the supported bus system can be used (Communication Client).



**Figure 26: Possible Internal Structure of a VAN Device DTM**

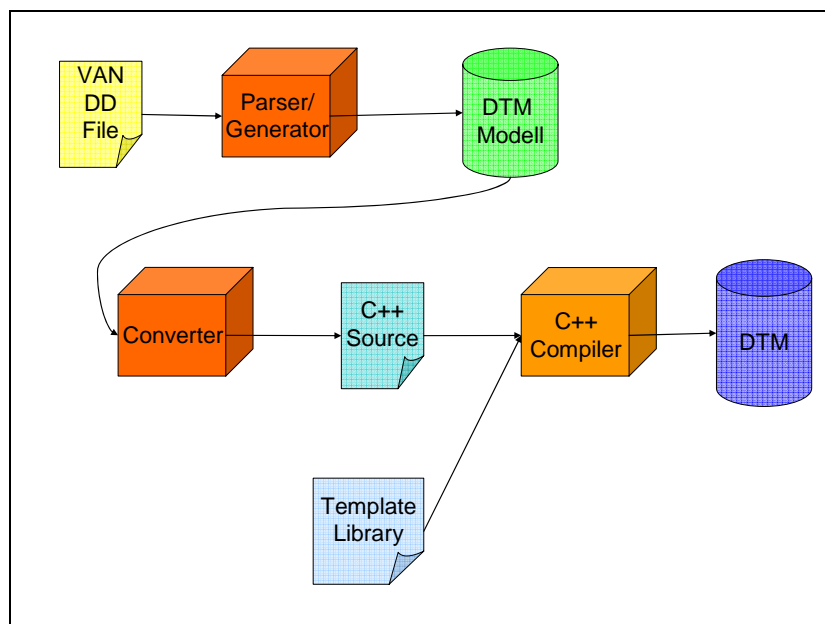
### 5.3.2 Adaptation of the VAN Device DTM to the VAN Device

Normally a DTM is dedicated to a specific device or supports a predefined device family. Figure 27 presents a discussed engineering environment, based on a FDT Frame Application. VAN Device Description could configure at runtime the VAN Device DTM inside the Device Tool, representing the VAN enabled device. All other runtime information about device address, etc will be passed via the TPF file (see chapter 4.3).



**Figure 27: Information Flow from Engineering Tool to FDT Frame Application**

In principle, two approaches for the adaptation of the VAN Device DTM to the VAN device are possible. The first one works at compile time of the VAN Device DTM, the second approach interprets the VAN Device Description at runtime. The tool chain of the first approach is shown in Figure 28. The VAN Device Description is provided as input to the generator. Therefore, a specific tool is necessary for parsing the Device Description in a first step. The output of this tool can either be an abstract model, if the further tool chain consists of such an abstraction layer or directly a source code, e.g. C++. This source code represents the class `Device DTM` of Figure 26, the `ActiveX` classes could be provided e.g. as Template Libraries. The sources, together with necessary libraries, will be processed by a standard compiler tool into the final VAN Device DTM.



**Figure 28: Generation of a DTM based on a VAN-DD**

In contrast to the prerequisite, that a DTM is dedicated to a specific device or device family, a VAN Device DTM have to be more generic. This means, it has to be adapted at runtime to the tangible

device type. For this purpose, the second approach can be used: the interpretation of the VAN-DD is made at runtime, e.g. at start up of the VAN Device DTM.

This approach of the interpretation requires a complete different implementation of the class `Parameter Handler`, shown in Figure 26. This class is additionally responsible for

- Reading the VAN-DD
- Establishing the relationships between the parameters
- Association of the described user interface inside VAN-DD with available `ActiveX` controls
- Activating the required user interface, depending on user action

Common for both approaches is the task of the VAN Device DTM, to read the provided TPF file, containing the runtime information of the device. This information will be provided by the VAN ECS and can therefore be used as default initialisation of the VAN Device DTM parameters.

### 5.4 VAN Communication DTM

A key factor to the success of the VAN project is its capability to have a smooth adaptation to the existing automation solutions and to conserve the technology investment already done from the companies. For this reason, it is important to be compliant with existing standards. Regarding the VAN Communication DTM the standard to be followed is the FDT interface specification 1.2.1 [FDT]. This interface specification is the latest one provided by the FDT Group. It is widely adopted by modern FDT based engineering tools since it provides several improvements of the version 1.2 which was previously used for most applications supporting the FDT/DTM technology.

This chapter provides a technical description of a communication DTM and it describes which parts of the FDT specification have to be taken into consideration during the development process of the VAN Communication DTM. Therefore, parts of this chapter are directly taken from the FDT specification.

Inside the architecture of the integrated concept for a VAN Engineering Tool (see chapter 2.2.2) an important role is covered by the VAN Communication DTM. The VAN Communication DTM must provide the communication functionality to the VAN Device DTM and acts as an interface between the FDT technology and the web service technology used for the ASE based communication with the VAN devices. These communication capabilities are provided by the VAN Communication DTM by means of a Communication-channel.

The Communication-channel provides interfaces and methods specified from the FDT Group. All interfaces for the Communication-channel defined as mandatory in the FDT specification have to be implemented in the VAN Communication DTM. The complete list can be found in chapter 5.4.2.

In general, the FDT architecture allows that the Communication-channels can be provided by the FDT Frame Application itself, by a Communication DTM or by a Gateway DTM. In the integrated concept for a VAN Engineering Tool the VAN Communication-channel is provided from the VAN Communication DTM. Nor a VAN Gateway neither the VAN Frame Application provide the VAN Communication-channel.

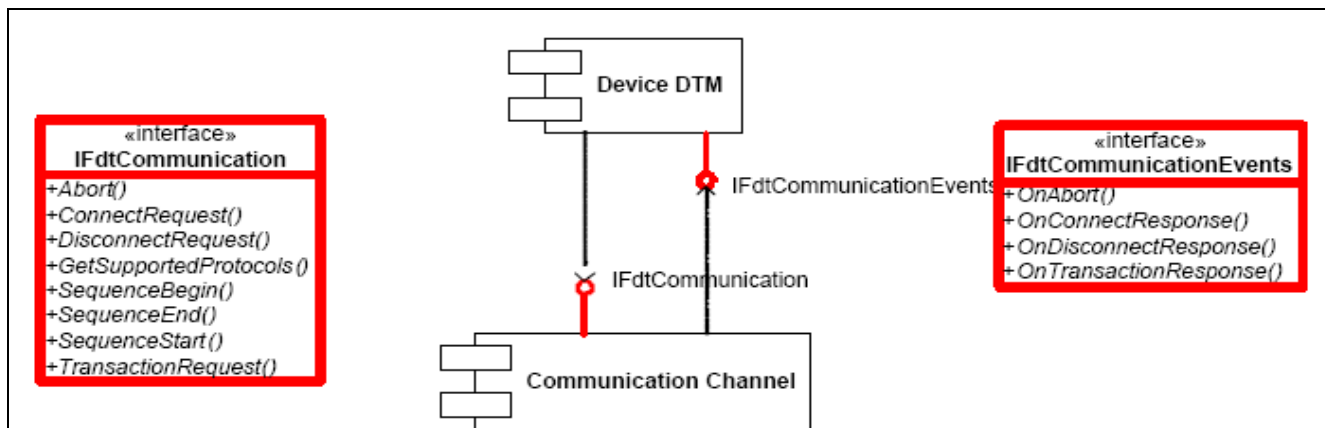


Figure 29: Basic Communication Interfaces

Figure 29 [FDT] shows the required interfaces between the Device DTM and the Communication-channel, *IFdtCommunication* and *IFdtCommunicationEvents*, as described in the FDT specification.

The *IFdtCommunication* interface allows a VAN Device DTM to ‘talk’ to its corresponding VAN device using the VAN Communication-channel integrated in the VAN Communication DTM. When the VAN Device DTM needs to communicate to its field device, it uses the *IFdtCommunication* interface and calls the methods *ConnectRequest()* and *TransactionRequest*. Afterwards, when the VAN Device DTM needs to stop the communication to the field device, the *DisconnectRequest* method of *IFdtCommunication* is called.

The *IFdtCommunicationEvents* interface is the counterpart to *IFdtCommunication*, which allows the Communication-channel to reply asynchronously to the VAN Device DTM's requests. Each call on *IFdtCommunication* receives a corresponding reply on *IFdtCommunicationEvents*. For example,

*OnConnectResponse* is replied to *ConnectRequest* and *OnTransactionResponse* is replied to *TransactionRequest*. The *IFdtCommunicationEvents* interface can also be used to verify if an online connection exists or not.

Information passed over VAN Device DTM and VAN Communication DTM interfaces are passed as XML documents. Each XML document must be in conformance to a defined fieldbus specific communication schema. The fieldbus communication schema integrates dedicated elements for each call. Therefore, a new VAN communication schema has to be specified for each method which is used in the communication between the VAN Device DTM and the VAN Communication DTM.

#### 5.4.1 Communication-channel Object Model

The core component of the Communication DTM is the Communication-channel. Therefore it is important to understand the structure of the Communication-channel and its interfaces.

Figure 30: FDT Object Model FdtChannel

[FDT] shows all the classes and methods that have to be implemented in the Communication-channel. As shown in

#### Figure 30: FDT Object Model FdtChannel

the VAN Device DTM communicates with the VAN device by sending requests to the VAN Communication DTM interface *IFdtCommunication* and receives responses by the interface *IFdtCommunicationEvents*. The remaining interfaces are described in chapter 5.4.2.

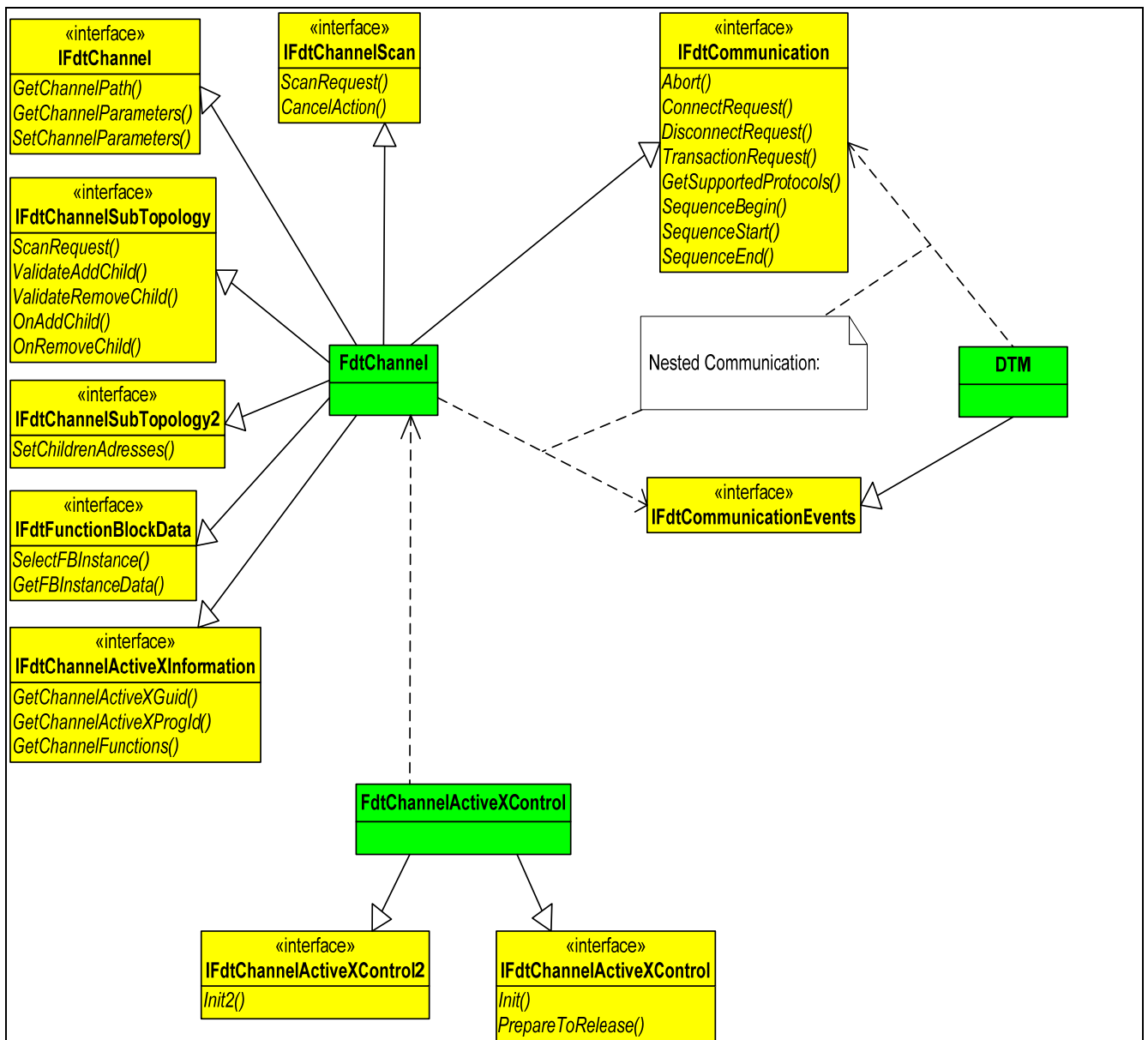


Figure 30: FDT Object Model FdtChannel

### 5.4.2 Communication DTM Interfaces

This chapter provides a complete list of the mandatory interfaces required for the VAN Communication DTM. The list was extracted from the FDT specification [FDT] in order to give the reader a quick overview.

The VAN Communication DTM must provide at the same time all the mandatory interfaces specified for the Device DTM as well as the mandatory interfaces specified for the FDT Communication-channel.

All these interfaces must be implemented in the VAN Communication DTM prototype. A complete analysis of possible simplifications for the implementation of the VAN Device DTM prototype will be provided in the deliverable D08.4-2.

To be compliant to the FDT specification all the following interfaces must be implemented.

- **IDtm:** This interface is the main interface of a DTM that supports FDT version 1.2 and older [FDT].

- **IDtm2**: This interface is the main interface of a DTM supporting FDT version 1.2.1 or higher versions. Via this interface such a DTM gets its initialization after the instantiation. This interface extends the interface *IDtm* by new methods. A FDT Frame Application supporting 1.2.1 or higher version has to use IDtm2, if the DTM supports this interface [FDT].
- **IDtmChannel**: This interface is used for accessing channel objects. On the one hand the supplied channel objects carry the information which is necessary to create the association between I/O channels of a device and the functions of the FDT Frame Application. On the other hand, in case of communication channels, these channel objects are used to build the communication chain for nested communication [FDT].
- **IDtmDocumentation**: This interface provides the DTM specific documentation for a device instance as XML document [FDT].
- **IDtmDiagnosis**: This interface provides the basic diagnosis functions required by a FDT Frame Application for DTMs with configuration parameters [FDT].
- **IDtmInformation**: This interface is the second main interface of a DTM according to FDT version 1.2 and older [FDT].
- **IDtmInformation2**: This interface extends the interface *IDtmInformation* by new methods defined in the FDT specification 1.2.1. Via this interface the DTM can be asked for its static information like version, vendor, and its capabilities to allow integration into the libraries of a FDT Frame Application [FDT].
- **IDtmOnlineDiagnosis**: This interface provides online diagnosis functions used by a FDT Frame Application to validate complete bus systems within a batch process [FDT].
- **IDtmParameter**: This interface allows a FDT Frame Application to access device parameters. The DTM provides its actual in-memory representation of its instance data set. It is up to a DTM and depends on the device-type and fieldbus-type which parameters are available [FDT].
- **IDtmSingleInstanceDataAccess**: This interface allows a FDT Frame Application the offline access to specific parameters of a device [FDT].
- **IFdtEvents**: This interface is the callback-interface for the FDT Frame Application [FDT] the behaviour of this interface is already described in chapter 0.
- **IFdtChannel**: This is the main interface of a channel that provides all information which is necessary for the channel assignment [FDT].
- **IFdtChannelSubTopology**: This interface allows validating the sub topology beneath a channel. A FDT Frame Application is responsible for the sub topology of a channel. It has to call this interface so that the channel or at least the according DTM can validate the configured connections [FDT].
- **IFdtChannelSubTopology2**: This interface has to be implemented by a Communication-channel and extends the interface *IFdtChannelSubTopology* by new methods. This interface supports network topology management functions for address setting. The interface is used to ask a VAN Communication-channel to set the fieldbus address of a single DTM or DTMs of a sub topology [FDT].
- **IFdtCommunication**: This interface is the communication entry point of a channel with communication functionality. A channel is able to support a number of different fieldbus protocols. Protocol specific XML documents are exchanged between a Communication-channel and a connected client via the *IFdtCommunication* and *IFdtCommunicationEvents* interfaces. The type of protocol to be used must be specified with a connect request [FDT].

- **IFdtFunctionBlockData:** This interface must be supported by DTMs for failsafe devices. The root communication component of an FDT system defines whether the system can provide failsafe access or not. In a system that is able to provide failsafe access, it is important that all communication components support failsafe access. Communication-channels have to support the propagation of the failsafe function calls. That is why the interface is mandatory for all channels with communication functionality and with gateway functionality [FDT].
- **IFdtChannelScan:** This interface defines methods, which replace scan related methods of the *IFdtChannelSubTopology* interface. If a Communication-channel supports FDT1.2.1 and runs in an environment that supports FDT1.2.1, the channel object can rely on that the methods of this interface are used instead of the replaced methods of the *IFdtChannelSubTopology* interface [FDT].

### 5.4.3 Roadmap for Developing a VAN Communication DTM

To realize a new Communication DTM inside the FDT/DTM technology a specific series of actions has to be taken. The following list shows the basic steps to realize a VAN Communication DTM.

- Step 1: A new VAN protocol ID for FDT/DTM has to be specified
- Step 2: Consider VAN device type information and formal device description
 

The formal device description contains data that is type specific and not instance specific. Examples of the described information are bus timing information or supported baud rates. The formal device description is provided by the DTM at *IDtmInformation*. A DTM of a fieldbus device provides this description during invocation of the method *GetInformation()*. This information is accessible via *deviceTypeInfoInformation*. An example for a device description is the Profibus GSD file.
- Step 3: Consider VAN instance information
 

The configuration strings providing VAN device instance specific information about the current configuration of the VAN device. The DTM provides the configuration string within the attribute *busMasterConfigurationPart* that is part of the XML document, which is available via *IDtmParameter* method *GetParameters()*.
- Step 4: Define VAN protocol specific schemas
 

The inclusion of the VAN protocol into the FDT specification causes a consideration of the data types used by the VAN protocol. Especially, it has to be checked if the data types of the VAN protocol uniquely map into the *FDTDataTypesSchema.xml* scheme. If there are inconsistencies then the FDT Working Group "Specification" has to be contacted, which maintains this schema.

The FDT specification contains realizations of this kind of schema in the appendix (e.g. for Profibus).
- Step 5: Define extensions related to FDT specification 1.2.1
 

A VAN protocol extension for the FDT specification 1.2.1 has to be provided. This step has to be done to make the VAN protocol part of the official FDT standard.
- Step 6: Prototyping
 

Prototyping is done in order to ensure a high quality of the newly defined specification. Therefore, it has to cover all protocol specific services defined in the VAN communication schema.

## Conclusion

The scope of this deliverable was the definition of the architecture of a VAN Engineering Tool and the identification of appropriate interfaces and technologies. Two different concepts for a VAN Engineering Tool have been presented in order to regard different requirements. On the one hand a stand-alone concept was introduced to satisfy the requirement to cover the VAN-ECS role as well as the VAN-ECS role by one tool and to provide an independent solution, which is can be used to supplement VAN features to existing engineering tools. On the other hand the integrated concept was proposed and elaborated to allow the use of standardized technologies and interfaces as FDT/DTM and TCI in order to add necessary VAN enhancements to existing engineering tools.

A first validation of the feasibility for the two concepts was done by means of a simplified use case and the investigation of concrete definitions for implementations of VAN Engineering Tool prototypes. For the stand-alone concept a component-based design for an Engineering GUI to be realized on the basis of the Eclipse platform was used. The prototype of the integrated concept was defined for the tools Step7 from Siemens and AutomationXplorer+ from Phoenix.

For both concepts of the VAN Engineering Tools the interfaces to the VAN-DD have been depicted and a concept for integration of the ASE objects and attributes in the VAN-DD was introduced. Also basic explanation of the FDT/DTM and TCI technology have been provided and the aspects relevant for the VAN Engineering Tool have been described.

In this deliverable several interfaces have been identified, which are used by a VAN Engineering Tool. In particular, each VAN Engineering Tool must be able to exchange information with the VAN Instance Model, to read a VAN-DD, and to download the configuration according to the ASE definition to the VAN device via web services. A VAN Engineering Tool following the integrated concept must be based on FDT/DTM technology or it must be able to use the TCI interface.

The technical specifications for these interfaces, the specification of the information to be exchanged via TCI, and a VAN communication schema for the VAN Device DTM and VAN Communication DTM have to be specified in the next deliverable D08.4-2 of task T8.4. Moreover, the VAN Information Model for the simplified use case has to be developed as an XMI file and a VAN-DD file has to be written before the implementation and the test of the prototypes can be completed in tasks T8.5, T8.6, and T8.7. In addition, the specifications for the ASEs to be implemented by the prototype have to be completed by work package WP2.

An implementation of the stand-alone concept is already started for the Engineering GUI in parallel to the specification work in task T8.4. The implementation of the integrated concept with the activities to enhance the tools Step7 and AutomationXplorer+ and to implement the VAN Communication DTM and VAN Device DTM has not yet started. An analysis has to be performed to define which components already developed in the Engineering GUI can be reused for the integrated prototype and the DTMs. The remaining activities for the prototype implementation have to be allocated to the different deliverables for the prototypes in tasks T8.5, T8.6, and T8.7.

# 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 Component	A group of VAN devices which realize a specific function. An automation component can be a single VAN device or it can comprise several VAN devices.
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 [TCI1].
Device	Any device used in an automation system, including network device.
Device Tool	The device specific tool implements a graphical user interface optimized for the requirements of the corresponding device and transfers data from and to the device [TCI1].
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.
Model View Controller	Model-View-Controller is the concept of encapsulating some data together with its processing (the model) and isolate it from the manipulation (the controller) and presentation (the view) part that has to be done on a user interface.
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.
System	Same as Automation System.

---

Subsystem	The term subsystem is used for a Profinet IO-System or a Profibus DP-Mastersystem. It is a set of IO Devices or DP-Slaves assigned to the same IO-Controller or DP-Master. Within a project of the Engineering System one or more subsystems can exist. Each DP-Master / IO-Controller owns exactly one subsystem [TC11].
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 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

API	Application programming interface
ASE	Application Service Element
DD	Device Description
DDXML	Device Description XML
DHCP	Dynamic Host Configuration Protocol
DTM	Device Type Manager
EDD	Electronic Device Description
ES	Engineering System (inside TCI specification)
FDCML	Field Device Configuration Markup Language
FDT	Field Device Tool
GSD	Generic Station Description
GSDML	General Station Description Markup Language
PID	Program Interface Description
TCI	Tool Calling Interface
TPF	Temporary parameter file
UI	User Interface
UML	Unified Modeling Language
UUID	Universal Unique Identifier
VAN	Virtual Automation Network
VAN-AD	VAN Automation Device
VAN-DD	VAN Device Description
VAN-ECS	VAN Engineering Client for Automation System
VAN-FS	VAN Function Set
VAN-PD	VAN Proxy Device
VAN-VD	VAN Virtual Device
XMI	XML Metadata Interchange
XML	Extensible Markup Language
WSDL	Web Service Description Language

## References

### References:

- [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
- [D02.2-3] Deliverable D02.2-3 of the VAN project. *Specification of the Open Platform for Automation Infrastructure*, The VAN Consortium, To be published in 2007
- [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, To be published in 2007
- [D08.1-1] Deliverable D08.1-1 of the VAN project. *Overview on existing engineering tools and Requirements for engineering tools for the "VAN" platform*. The VAN Consortium, 2006
- [D08.2-1] Deliverable D08.2-1 of the VAN project. *Engineering process concept and specification*. 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, 2006
- [ECLIPSE] Web site of Eclipse platform (<http://www.eclipse.org/platform>)
- [FDT] FDT Interface Specification, Version 1.2.1, March 2005
- [FDT Group] Web site of FDT Group (<http://www.fdtgroup.org>)
- [FDT\_Tec] FDT Technical Description, September 2006
- [HadTNC] T. Hadlich, T. Szczepanski: *FDT - The new concept for fieldbus communication*. In Control Engineering Europe, September 2002 ([http://www.ifak-system.com/information/CEE\\_FDT.pdf](http://www.ifak-system.com/information/CEE_FDT.pdf))
- [IEC61158] International Electrotechnical Commission: IEC 61158 *Digital data communications for measurement and control - Fieldbus for use in industrial control systems*. 2003
- [ISO15745] International Standard Organisation: ISO 15745 *Industrial automation systems and integration - Open systems application integration framework*. 2003
- [Step7Int] SIMATIC *Erste Schritte und Übungen mit Step7*, Siemens
- [TCI1] PROFIBUS PROFINET Specification: *Tool Calling Interface*, Version 1.0, October 2006, Order No: 2.602
- [XMI] OMG XML Metadata Interchange (XMI) Specification, V1.2, January 2002 (<http://homepages.inf.ed.ac.uk/perdita/XMI>)