



VAN

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

Virtual Automation Networks

Work Package 8

Engineering of VAN platform for Embedded Automation Systems

Task 8.6

Engineering Tool Prototypes for Integrated Concept

Deliverable D08.6-1

Specification of Engineering Tool Prototypes for Integrated Concept

Document type	: Deliverable
Document version	: Final
Document Preparation Date	: 30.04.2008
Classification	: Confidential
Contract Start Date	: 01.09.2005
Duration	: 31.08.2009



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

Rev.	Content	Resp. Partner	Date
0.1	Document structure	Deuter, Phoenix Contact	07/10/01
0.2	Review Final Version	all	08/04/04
0.3	Merging according to workshop results	all	08/04/16
0.9	Final Draft	all	08/04/24
1.0	Final	Phoenix	08/04/28

Final approval	Name	Partner
Review Task Level	Gerhard Freitag	Phoenix
Review WP Level	Friederich Götz	Schneider
Review Board Level	Axel Klostermeyer	Siemens

Executive summary

This document is the deliverable D08.6-1 of the VAN project and comprises the first results from the activity of the participants within task T8.6 “Engineering Tool Prototypes for Integrated Concept” of work package WP8 “Engineering of VAN platform for Automation Systems”. It is a result of cooperation of the parties: ifak e.v. Magdeburg (ifak), Schneider Electric (Schneider), Siemens AG (Siemens) and Phoenix Contact (Phoenix).

This deliverable starts with the chapter *Introduction*, which gives a short outline about the basic results of the previous tasks, which are necessary for a successful realisation of a prototype for an engineering tool following the Integrated Concept. Moreover, it provides a short overview about the main topics of this document.

The chapter *System Architecture for VAN Integrated Prototype* gives a global overview about the components used in the Integrated Prototype. There are many components described in detail in the further chapters.

The chapter *Automation Example* includes the use case on which the specification is based. The same example is used for work package WP5 of the VAN project. This shows the interoperation between the work packages inside the VAN project.

The chapter *VAN STEP7 Integration* specifies how the tool Siemens STEP7 (STEP7) is enhanced in order to deal with VAN functionality. Here, one way to integrate VAN in already existing tools is shown. STEP7 makes the TCI call to the engineering tool Phoenix AutomationXplorer+ (AutomationXplorer+). The role of STEP7 in this Integrated Concept of VAN engineering is explained.

The chapter *VAN AutomationXplorer+ Integration* explains the role of AutomationXplorer+ in this concept. On one hand, it receives a TCI call from STEP7 with engineering data and on the other hand it works as FDT frame application and instantiates the VAN Device Type Managers (DTM), which are used for device integration.

The chapter *VAN Communication DTM* describes the DTM, which builds up the connection to the VAN device. This is a new implementation, so first the architecture of this component is explained. Afterwards, the VAN communication schema, a tool to generate the VAN communication schema and also the implementation of Field Device Type (FDT) interfaces and of the Web Service Client are described.

The chapter *VAN Device DTM* describes the DTM which is responsible for the configuration of the VAN device via ASE parameters. So here is an overview about the components inside this DTM shown and the implementation of the FDT interfaces is described.

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

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

Contents

1	Introduction	8
2	System Architecture for VAN Integrated Prototype	9
3	Automation Example	10
4	VAN STEP7 Integration	11
4.1	Import a VAN-DD Instance File	13
4.2	Detailed VAN Engineering in the Local Tool	15
4.3	TCI Call by STEP7	17
5	VAN AutomationXplorer+ Integration	19
5.1	Enable AutomationXplorer+ as TCI Device Tool	19
5.1.1	Provision of the PID File	19
5.1.2	Detailed Description of the Registration Process	20
5.2	Start-up Phase in TCI Mode	22
5.2.1	Start-up behaviour of AutomationXplorer+ in TCI Mode	22
5.2.2	Decode TPF File	22
5.3	Instantiation of DTMs and Transfer of Information	23
5.4	Implementation of FDT Interfaces	23
6	VAN Communication DTM	27
6.1	Architecture of VAN Communication DTM	27
6.1.1	General Overview of VAN Communication DTM	27
6.1.2	Internal Overview of VAN Communication DTM	29
6.1.3	VAN Communication DTM Interactions	31
6.1.4	Bus Category Identifier	31
6.2	VAN Communication Schema	32
6.2.1	Schema Transformation Algorithm	33
6.2.2	VAN Communication Schema Generator	35
	VAN CSGen - User Interface Description	38
	VAN CSGen - How to Use the Tool	39
	VAN CSGen - Components Behaviour	40
6.3	Implementation of FDT/DTM Interfaces	41
6.4	Implementation of Web Service Client	46
6.4.1	Web Service Code Generators	46
	gSOAP 46	
	.NET 46	
	Apache Axis	46
	Dynamic Web Service Client	46
6.4.2	Web Service Client Prototype Implementation	47
7	VAN Device DTM	48
7.1	Overview about components	48
7.2	Implementation of FDT interfaces	50

7.2.1	Device DTM.....	50
7.2.2	Control for Dtm	52
7.2.3	Communication DTM.....	53
7.2.4	Frame Application.....	53
8	Conclusion.....	54
	Glossary.....	55
	Terminology	55
	Abbreviations.....	57
	References.....	58

List of Figures

Figure 1: Overview of Components.....	9
Figure 2: Automation Example	10
Figure 3: Example Virtual Automation Network.....	11
Figure 4: Example Stand-Alone + Integrated Tool	12
Figure 5: Universal Connection	13
Figure 6: Instantiation of a VAN device in the supervision station	14
Figure 7: Show and Edit of a ASE.....	15
Figure 8: Send and Receive PROFINET IO VAN concept to different locations.....	16
Figure 9: Web-interface to show the operation	17
Figure 10: Start the device tool with engineered parameters.....	18
Figure 11: Value TciExtraData	18
Figure 12: Registration Process part 1.....	20
Figure 13: Registration Process part 2.....	21
Figure 14: Example for devices and tools in the TCI section of the Windows registry.....	22
Figure 15: General Overview of the VAN Communication DTM	27
Figure 16: Class Diagram of VAN Communication DTM	29
Figure 17: Behaviour Diagram of VAN Communication DTM	31
Figure 18: Dependencies of Documents and Components.....	32
Figure 19: VAN CSGen Architecture.....	35
Figure 20: VAN Communication Schema Generator Class Diagram.....	36
Figure 21: VAN CSGen Main Window	38
Figure 22: VAN CSGen Menu Options.....	39
Figure 23: VAN CSGen Behaviour.....	40
Figure 24: VAN Communication DTM Interfaces Usage.....	41
Figure 25: VAN Communication DTM Interfaces - Level of Implementation.....	42
Figure 26: VAN Device DTM Architecture.....	48

List of Tables

Table 1: Mapping of Instance Information	23
Table 2: Description of Implementation Levels	23
Table 3: Implementation Levels for the Frame Application Interfaces	24
Table 6: VAN Communication DTM Interfaces	42
Table 7: Methods of <code>IPrm</code>	49
Table 8: Methods of <code>IVanDtm</code>	49
Table 9: Methods of <code>IVdp</code>	50
Table 10: Level of Implementation for Device DTM Interfaces	50
Table 11: Level of Implementation for Control the DTM.....	52

1 Introduction

The objective of task T8.6 is to specify a prototype for an engineering tool following the Integrated Concept. This specification is based on the previous tasks in work package WP8, where an engineering concept for Virtual Automation Networks (VANs) has been worked out and two feasible engineering architectures have been identified.

One of these two architectures is the Stand-Alone Concept, which follows the idea to provide a vendor independent solution regarding existing tools. Thus, it can be used for the development of a new tool, which covers the role of a VAN Engineering Client for Automation Systems (VAN-ECS) for the engineering of the whole VAN system as well as the role of a VAN Engineering Client for Automation Devices (VAN-ECD) for the configuration of a VAN device. The VAN-ECS and the VAN-ECD are introduced in [D08.2-1] in more detail. The architecture of this concept is component-based to ensure reusability for other engineering tools. The specification and implementation for a prototype of the Stand-Alone Concept is done in task T8.5, which is executed in parallel to task T8.6.

The other architecture is the Integrated Concept, which concentrates on the integration of necessary VAN enhancements into existing engineering tools. Therefore, well established engineering tools as well as the standardised interface technologies FDT/DTM and TCI are selected and used. Further, the UML-based VAN Information Model - that is introduced in [D08.3-1] - is used for the engineering of the network capabilities of a VAN.

The design for the Stand-alone Concept and the Integrated Concept have been synchronised in order to allow high level of reusability for the components. To avoid redundant information in the deliverables, all reusable components are described only in deliverable [D08.5-1], although many of these components are also used for the Integrated Concept described in the deliverable at hand.

During the execution of task T8.6, two deliverables have to be worked out. The focus of the first deliverable is the specification of an engineering prototype based on the Integrated Concept described in [D08.4-2]. The main task during the second phase of task T8.6 will be the implementation of the Integrated Prototype. Therefore, the second deliverable will contain the documentation of the implemented solution.

In this deliverable the Integrated Tool is specified. As mentioned above, standardised interface technologies are used. The solution is based on components which are either existing tools or new developed components. An overview of these components and interfaces is given to show, how these parts work together.

To make this specification more concrete and close to the implementation which will be done in the next phase of task T8.6, an Automation Example is introduced. The same example is used for work package WP5 of the VAN project and the specification of this Integrated Tool is done with this example in mind. So, the engineering will have close interfaces to other WPs to support their requirements.

All components are described and their interfaces are specified in detail. A description, which parts of the standardised interface technologies FDT/DTM and TCI are implemented for the prototype, is included. Also a description of the architecture and functionality of the new designed components is included.

Moreover, it should be mentioned, that the functionalities that will be implemented in the Integrated Tool during the second phase of task T8.6 have a strong relation to the requirements of VAN. Therefore, a questionnaire was worked out and distributed to all work packages. This ensures that no important aspect get lost during specification and implementation. Thus, the feedback is considered for the Integrated Tool.

2 System Architecture for VAN Integrated Prototype

The Integrated Prototype is the prototype implementation of VAN Engineering Client following the Integrated Concept as introduced in [D08.4-1] and described in [D08.4-2]

The Integrated Prototype enables existing tools to deal with VAN functionality. In the Figure “Overview of Components” the components of the VAN Integrated Prototype are shown in a global overview.

The existing tools STEP7 [STEP7] and AutomationXplorer+ [AX+] are enabled for VAN in this Integrated Prototype. Between these tools, the standardized TCI (Tool Calling Interface) [TCI1] is used. The communication and configuration of the VAN devices is done by using the FDT [FDT1.2.1] technology.

A detailed description for each of the components is placed in further chapters of this document.

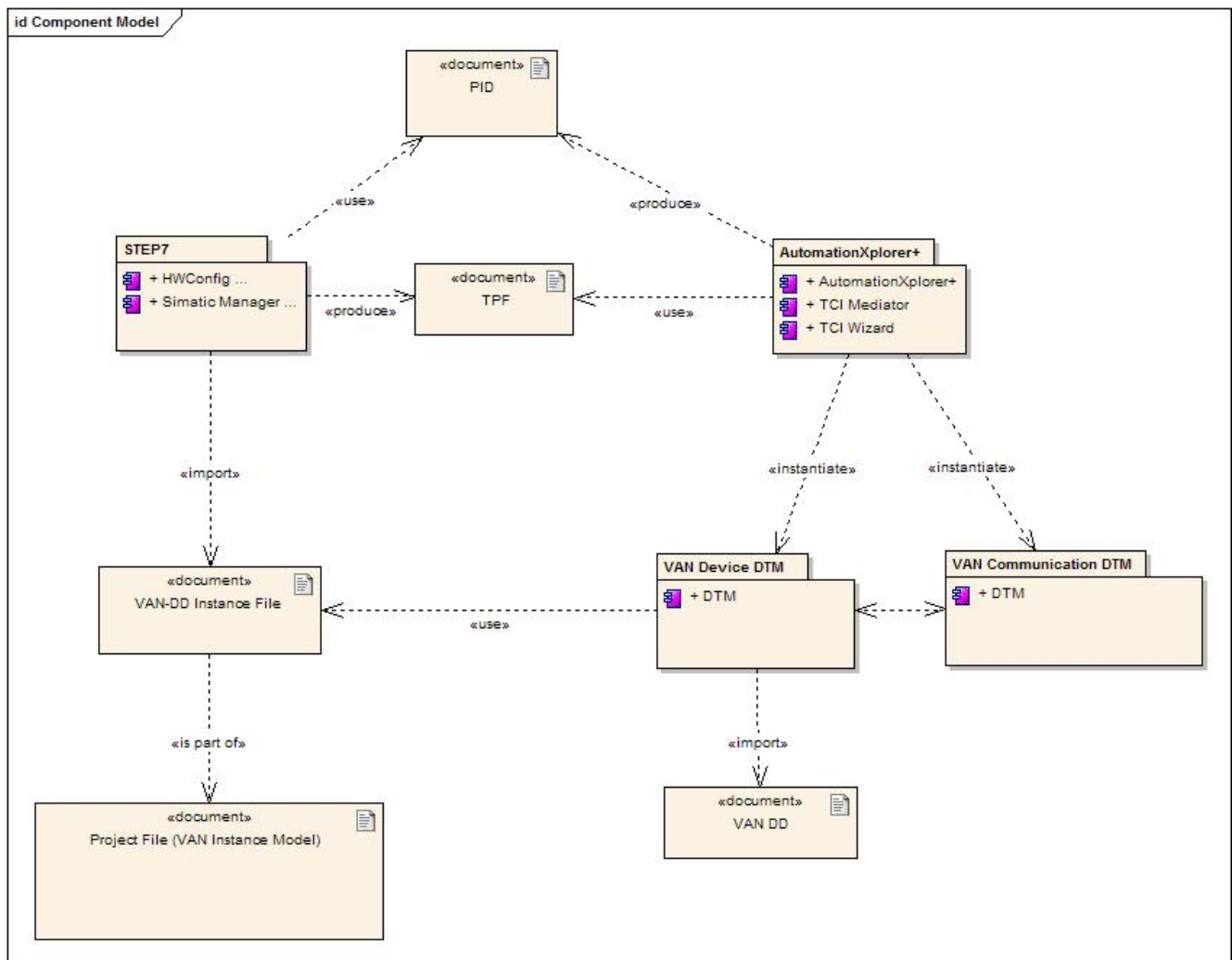


Figure 1: Overview of Components

3 Automation Example

For the specification of the Integrated Prototype of the VAN Engineering Client, the setup as described in [D05.4-1] is used. This example is also shown in the figure below.

This Automation Example consists of PROFINET IO Controller, PROFINET IO DEVICE and two Personal Computers (PC). The two PCs are used as VAN devices and the secure OpenVPN tunnel between these devices is configured via ASEs from the VAN Engineering Client.

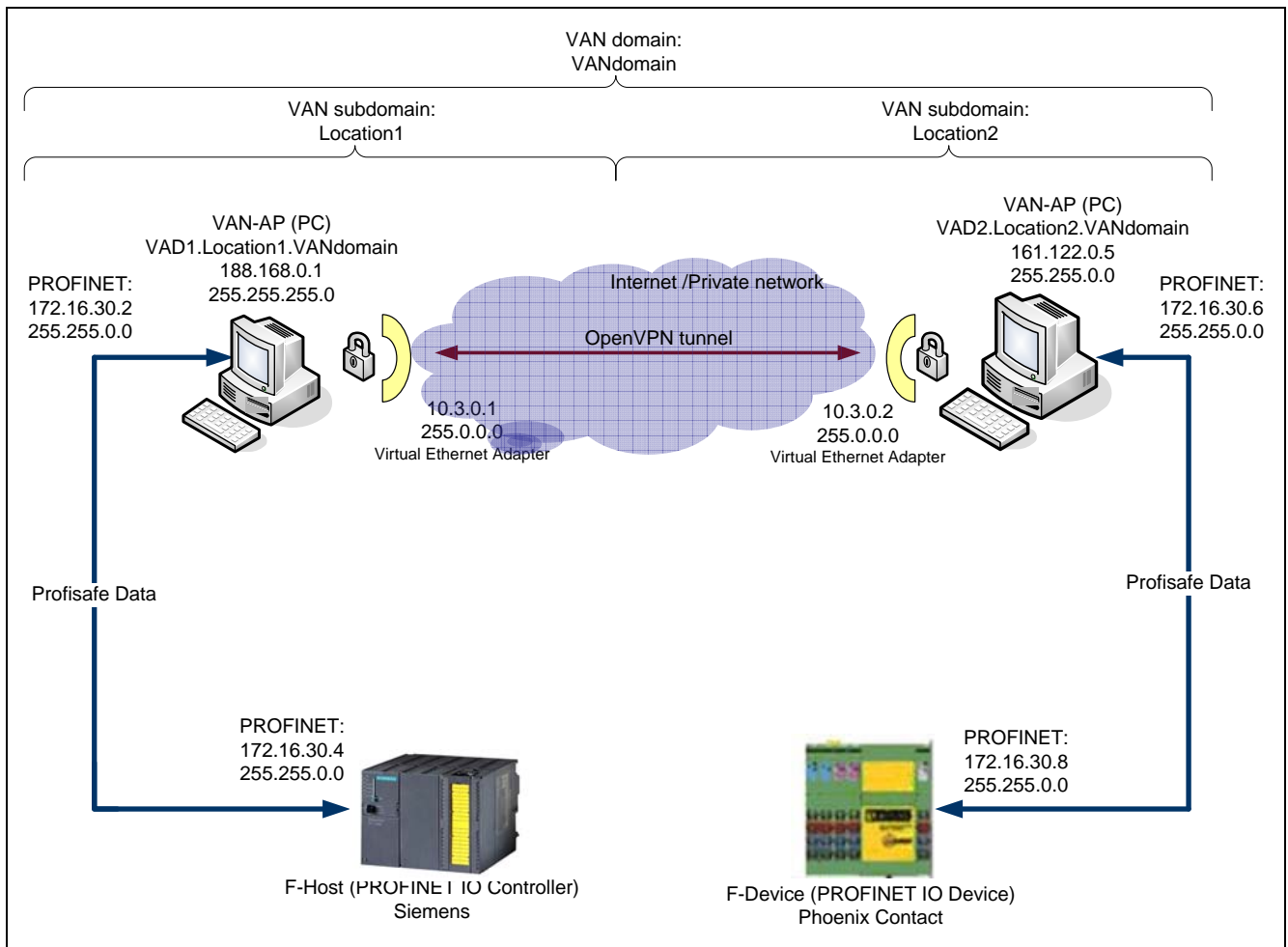


Figure 2: Automation Example

4 VAN STEP7 Integration

STEP7 supports the integration of VAN devices and imports the VAN-DD Instance File, in order to allow the user to edit the ASE parameters, and send the VAN engineered parameter to the Device Tool with help of the TCI standard as described in [D08.4-2]. The VAN-DD Instance File is a separate part of the VAN Project File as described in [D08.5-1]. The Integrated Tool is the combination of STEP7 and AutomationXplorer+, which are both using the TCI standard. With Using STEP7, which acts as Engineering System the user can define the access to the local industrial communication and the Device Tool AutomationXplorer+ configures the VAN devices, which builds the tunnel infrastructure through a public network.

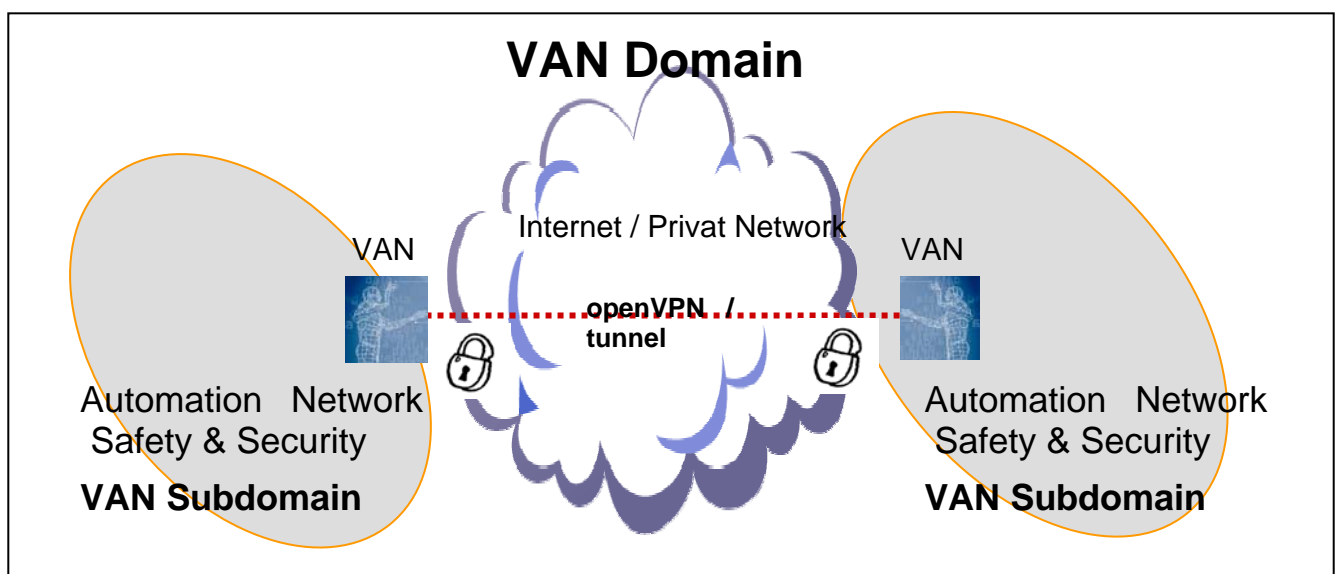


Figure 3: Example Virtual Automation Network

The Automation Example described in chapter 3 has only one tunnel and can also be configured with this concept. In the following example is a supervision station, which points with three VAN tunnels through a public network to three different plants. First the Integrated Tool imports the VAN-DD Instance File from the Stand-Alone Tool. Secondly the Integrated Tool calls after local engineering the Device Tool with the TCI standard. Now the Device Tool configures with web services the VAN Domains. And at the final end, the complete VAN engineered devices are stored in the VAN Project File.

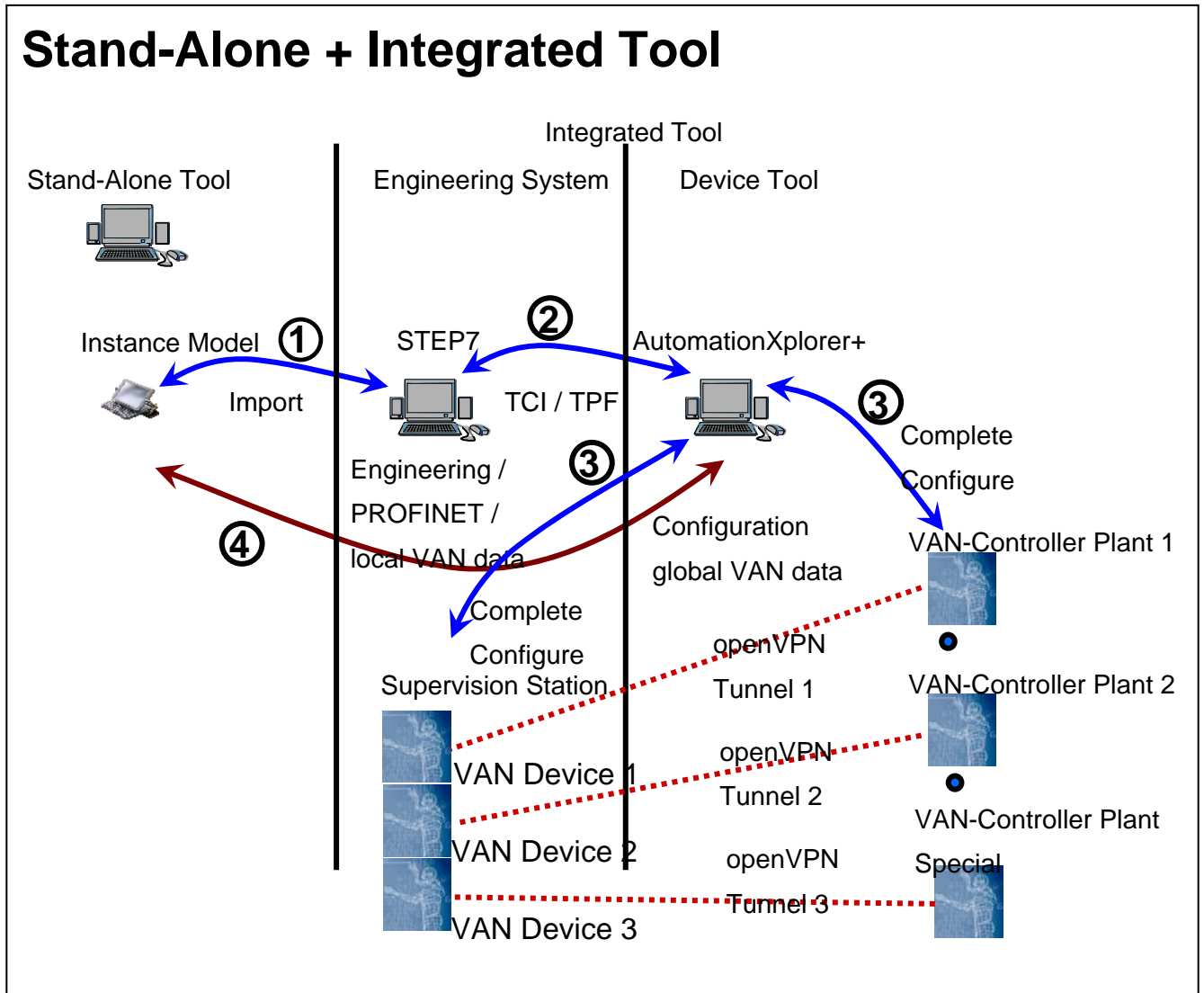


Figure 4: Example Stand-Alone + Integrated Tool

The Norm Devices for PROFINET or PROFIBUS are implementations of the standard defined by the PNO and have GSD files. Information could be found at [PI]. Because of the use of GSD in combination with a PROFINET Norm Device as VAN device all properties of PROFINET-IO for safety and security to the VAN device are passed. (see the possibility of security through openVPN in chapter 3). The security is not only possible for the tunnel through openVPN, it is also possible to make safety and security for the transported automation medium PROFINET in the tunnel.

A STEP7 Norm Device handles all devices derived from GSD files and has the properties of a PROFINET device. There exist two important Norm Devices in STEP7, one is for PROFIBUS and one for PROFINET. Now the third prototype Norm Device is a VAN device, which is derived and similar defined like the PROFINET Norm Device. Now all properties of the VAN device could be propagated to all devices which are also derived from a PROFINET VAN Norm Device. Now the VAN device can be connected by transfermodules as described in [D08.4-2] to any GSD-based PROFINET IO device. That means that any IO-data which is coming from any GSD-based module can be sent or received via PROFINET through a VAN device. The PROFINET IP-telegrams are transported through a VAN Tunnel where the start- and the endpoint are defined in the VAN-DD Instance File. (see also [D08.5-1])

All not from a GSD derived devices are not supported by the object model of STEP7 for the use with transfermodules, but can be connected with a universal transfer connection relation.

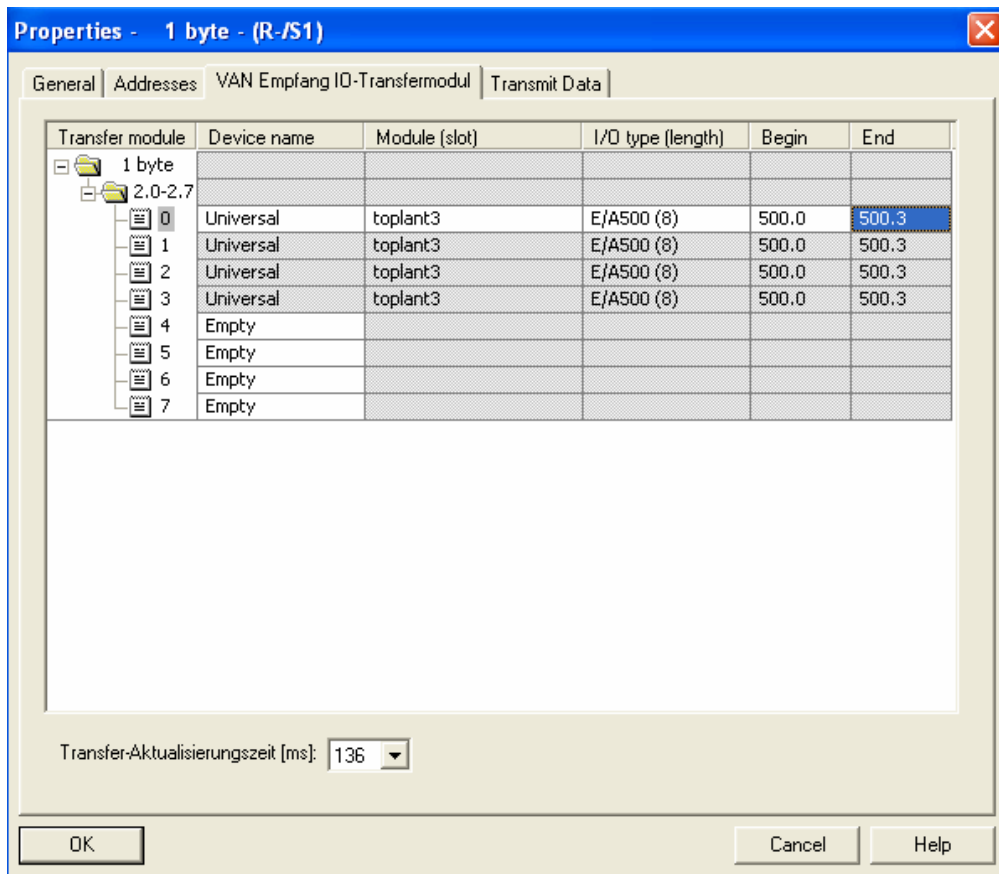


Figure 5: Universal Connection

The universal connection is done by addressing the remote CP by accessing the memory of the processor directly.

4.1 Import a VAN-DD Instance File

If a VAN device is placed in the local engineering tool from the hardware catalogue of STEP7 to the engineering view of [HW-Konfig], which is a part of the Simatic Manager, than the VAN device is automatically connected to the PROFINET IO Ethernet-System. (Industrial Ethernet: PROFINET-IO-System)

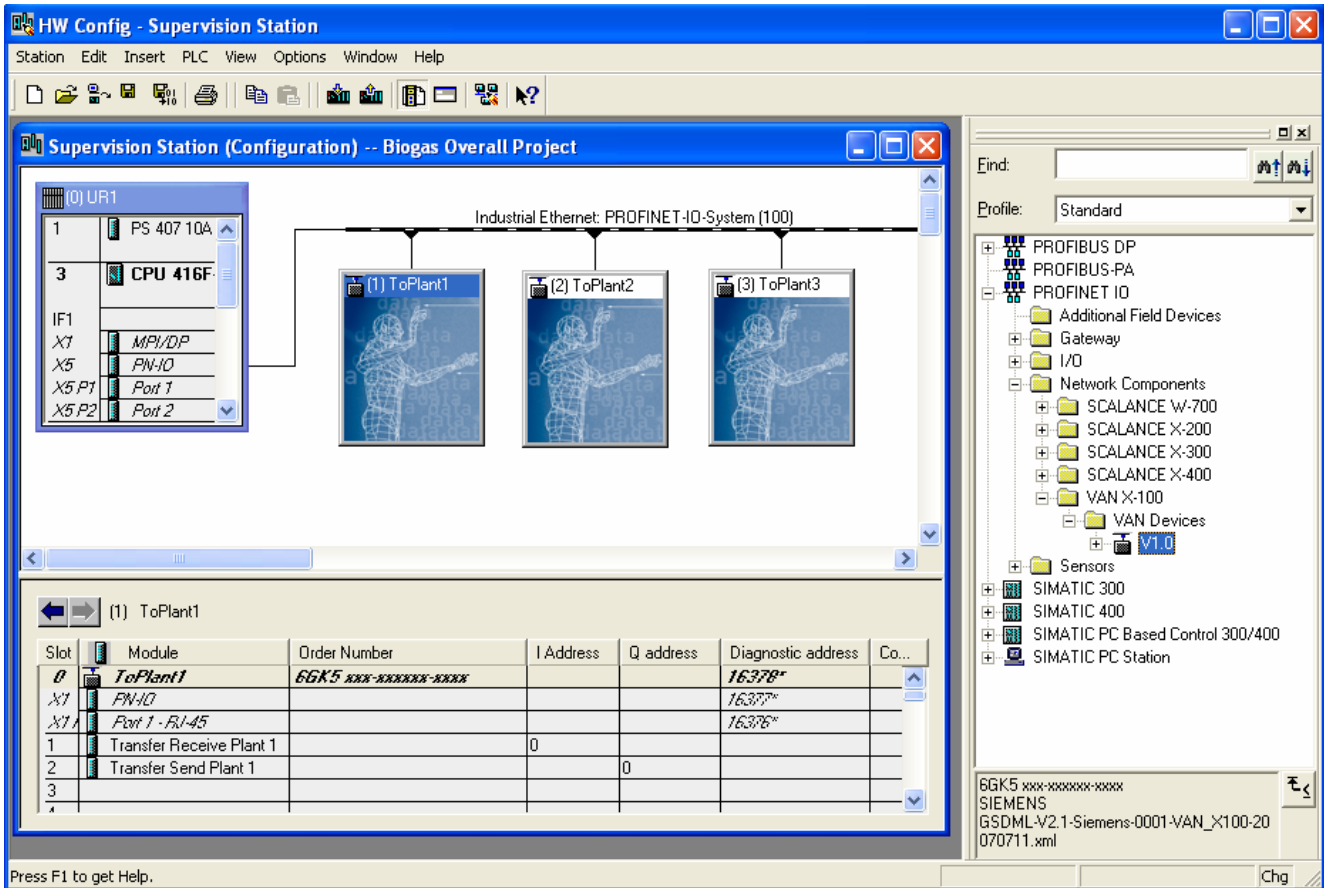


Figure 6: Instantiation of a VAN device in the Supervision Station

The supervision station is a place, where in our example is the endpoint of one or more VAN tunnels to different plants. Because the VAN device must have a unique name in the local STEP7 project, the STEP7 can not take the name from the VAN Instance Model without to ask the STEP7 naming server which is integral part of the engineering tool STEP7. If the VAN Name of the device is accepted from STEP7 as unique for the local engineering tool point of view, the corresponding ASEs for the VAN device are found. Now it is possible to show the ASEs and to edit some parameters for engineering the VAN device. For the prototype we edit the IP-address and the VAN name.

4.2 Detailed VAN Engineering in the Local Tool

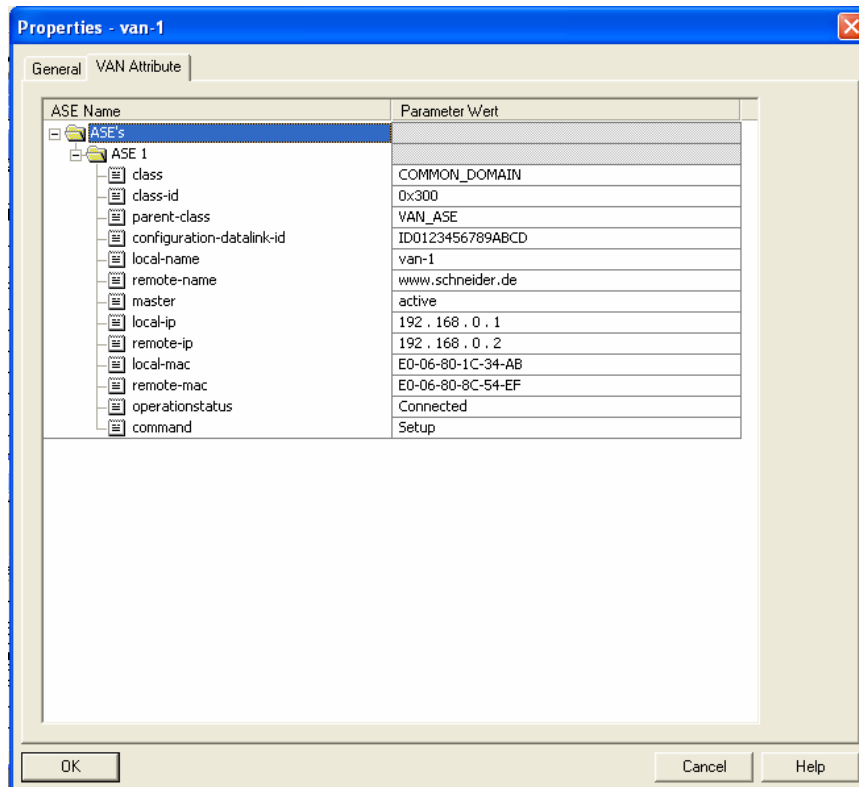


Figure 7: Show and Edit of a ASE

Through the VAN Instance Model all necessary ASEs are imported and shown. The engineer has the possibility to edit detailed information for building up of the tunnel to send or receive via PROFINET. There could be build up more than one tunnel through different VAN devices. Several ASEs could be responsible for a VAN device.

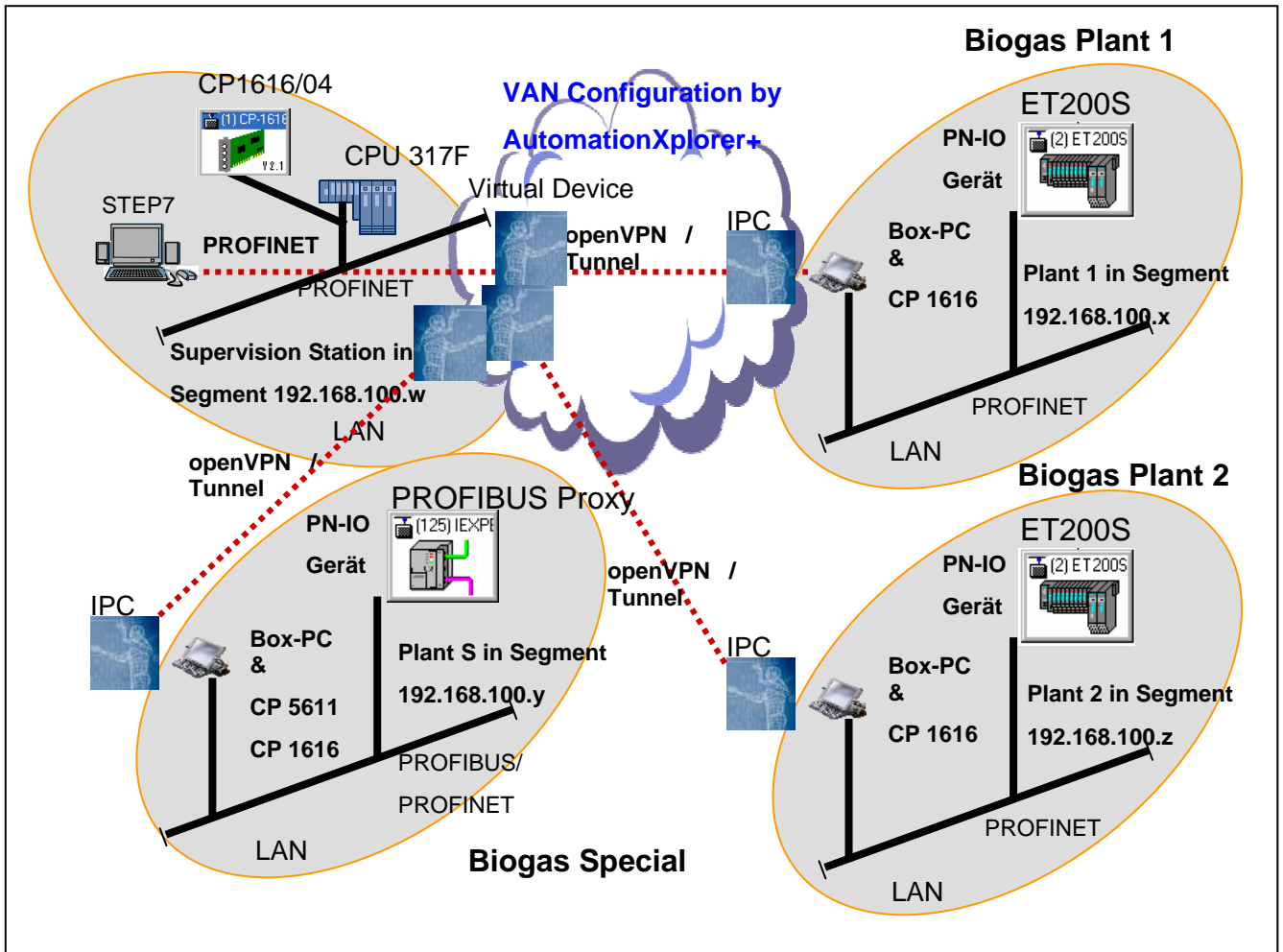


Figure 8: Send and Receive PROFINET IO VAN Concept to Different Locations

The VAN Instance Model has the possibility to define and collect the different endpoints to one or more tunnels. In the project file of the supervision station, each VAN device is represented by a GSD-based device, which is virtual and does also produce system data blocks for the CPU of the supervision station for the IO-routing and one system data block for the firmware of the CP controller at the plant side. The IO-routing is configured between a CPU and several controllers so called CPs, where the controllers are sending PROFINET telegrams to the CPU at supervision station side. The physical controller has therefore two functions, firstly to act as controller CP and secondly to act as virtual device at CPU side. If everything is engineered and configured right, than in our example a web-interface shows the operation as controller and the operation as device (data exchange).



Figure 9: Web-interface to Show the Operation

4.3 TCI Call by STEP7

All the parameters edited by the engineer of the tool STEP7 can be saved back the responsible part of the instance file to ensure, that the additional information of the engineering tool is not lost, which the user engineered. Now the VAN Engineering for the VAN devices is done. The engineer now can use the TCI mechanism (also described in [D08.4.2]) to start the Device Tool with the necessary information for configuration of the VAN device.

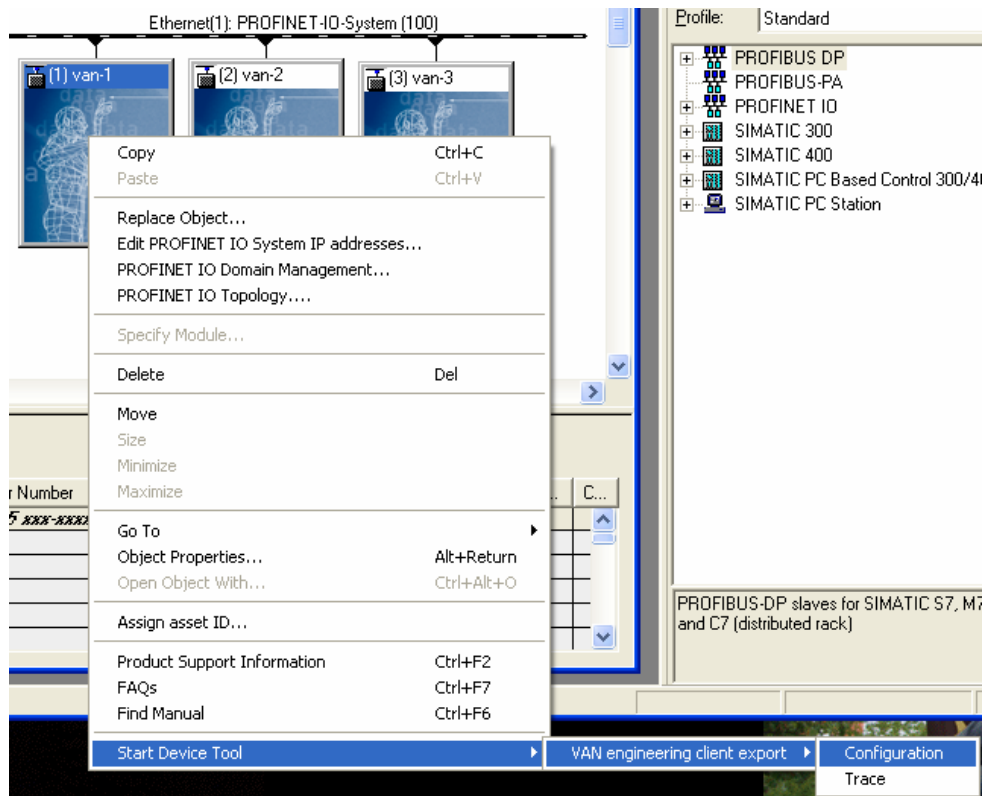


Figure 10: Start the device tool with engineered parameters

The TCI is a standard defined by the [PI] Profinet International. For the prototype we defined an extra value TciExtraData, which can send the freely engineered data to the Device Tool with TCI. For the prototype we send the name of the VAN-DD Instance File, that the Device Tool can find the corresponding ASEs.

```
Name="UsedConfigFile" |
Value="D:\Siemens\Step7\S7data\gsd\GSDML-v2.1-Siemens-0001-VAN_X100-20070711.xml"/><ParameterItem
Name="TciExtraData"
Value="636C61737300434F4D4D4F4E5F444F4D41494E00636C6173732D696400307833303000706172656E742D636C61737
30056414E5F41534500636F6E66696775726174696F6E2D646174616C696E6B2D69640049443031323334353637383941424
344006C6F63616C2D6E616D650076616E2D310072656D6F74652D6E616D65007777772E7363686E65696465722E6465006D6
17374657200616374697665006C6F63616C2D697000313932202E20313638202E2030202E20310072656D6F74652D6970003
13932202E20313638202E2030202E2032006C6F63616C2D6D61630045302D30362D38302D31432D33342D41420072656D6F7
4652D6D61630045302D30362D38302D38432D35342D4546006F7065726174696F6E73746174757300436F6E6E65637465640
0636F6D6D616E6400536574757000"/></ParameterList><ModuleList><ModuleItem Name=" 1 byte"
```

Figure 11: Value TciExtraData

5 VAN AutomationXplorer+ Integration

5.1 Enable AutomationXplorer+ as TCI Device Tool

5.1.1 Provision of the PID File

By the fact that the AutomationXplorer+ [AX+] is a FDT frame application; the tool can be used to work with potentially all devices that come with a DTM. The FDT frame application can host multiple DTM instances of various types and various device types. On the other side, the STEP7 [STEP7] hardware configuration tool can be used to setup a complex network with multiple attached devices of various devices types as well. STEP7 provides functionality to configure and parameterize networks, especially network controllers and attached devices. In addition to that the FDT frame application in connection with the DTMs can provide more detailed configuration, parameterization and diagnostics at the level of the attached devices.

The integration and invocation of the device tool like the FDT frame application AutomationXplorer+ into STEP7 is controlled by the content of PID file.

The PID file controls how the device tool appears within the STEP7 context menu.

```
<ToolDescription xml:lang="en" ToolName="AutomationXplorer+"
  ToolDescription="FDT Device Configuration Tool"/>
```

Furthermore the PID file specifies the command line parameter that is required in connection with the TPF file to invoke the device tool in TCI mode.

```
<InvocationPrefix Name="-i"/>
```

To achieve a seamless integration that includes the synchronization of the relevant parts of the topology, two optional TCI features are used. As described above, both software tools can deal with multiple device instances and types. So the PID file specifies the feature "UseMultipleDevice-Information", which is available in the TCI conformance classes 2 and 3 [TCI1][TCI1.1]. Additionally, the AutomationXplorer+ supports the synchronization in two ways: adding and deleting objects. This functionality is represented by the feature "SupportsObjectDeletion".

```
<ConformanceClass Name="C3">
  <OptionalFeature Name="UsesMultipleDeviceInformation"/>
  <OptionalFeature Name="SupportsObjectDeletion"/>
</ConformanceClass>
```

The location of the PID file and the location of the device tool are stored in the Windows registry. Static files associated with software are located in the Program Files folder as recommended in the Microsoft Windows Design Guidelines: <ProgramFiles>\<VendorName>\<ProductName>

The exact location depends on the used Windows Version (Windows 2000, Windows XP or Windows Vista) and Language Version (localized or multi-lingual).

Dynamically created files that should be accessible by all users independent of user privileges are located in the user profile: <AllUsersProfile>\<ApplicationData>\<VendorName>\<ProductName>.

The PID file is created during the registration process as described in chapter 5.1.2. So the PID file will be created within the profile folder that is common to all users.

5.1.2 Detailed Description of the Registration Process

The DTM catalogue handling is split into two different stages. The first stage realizes the standard FDT mechanism to set up a DTM catalogue through performing a registry scan. After the scan all found DTMs are instantiated to retrieve the device type information. The found device types can be added to the DTM catalogue by the user. The first part of this process is typical for all FDT frame applications.

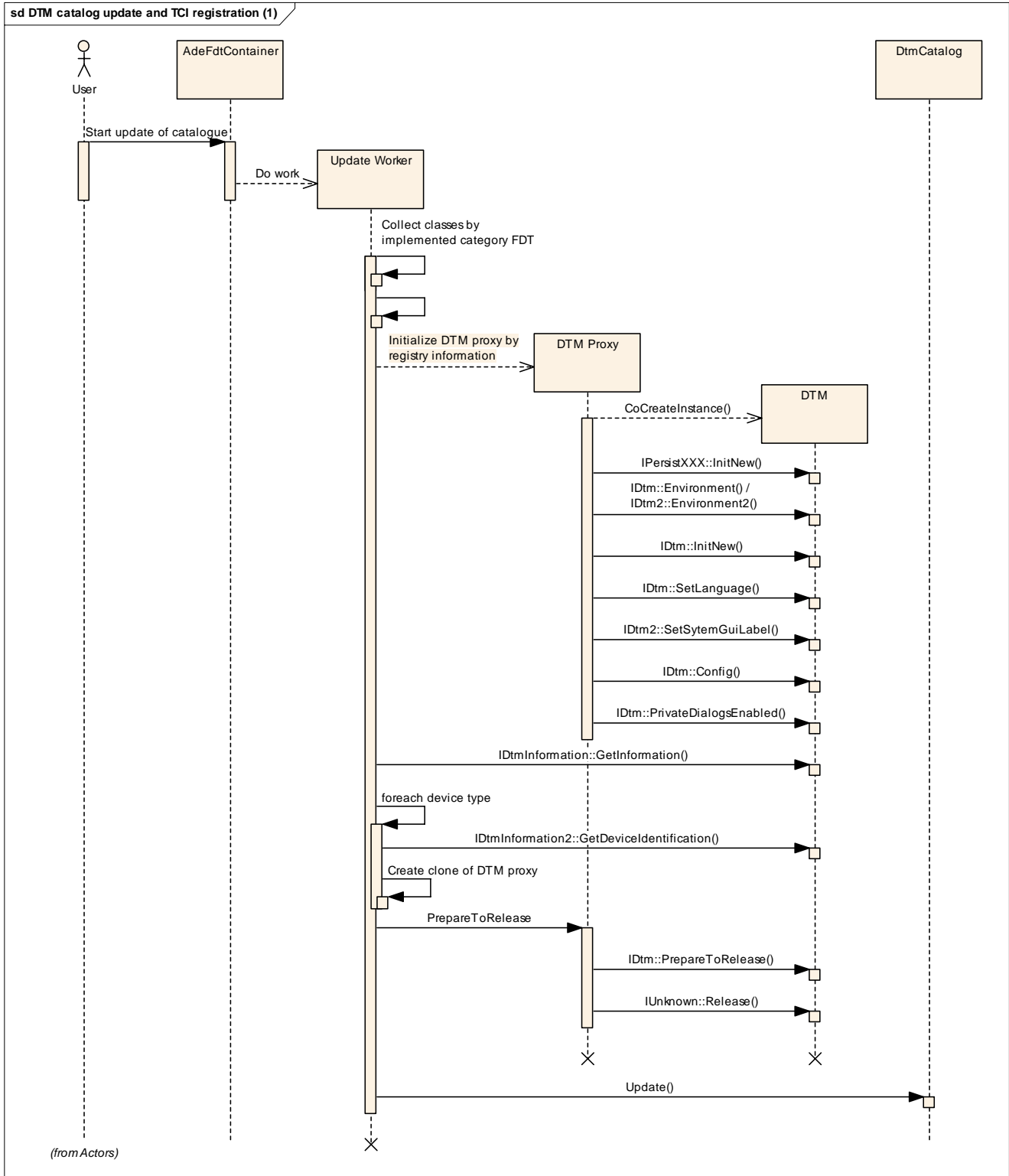


Figure 12: Registration Process part 1

At the second stage the combinations of manufacturer ID and device type ID are added to the registry. Both identifiers are mandatory attributes in the device type information for several bus

protocols, e.g. PROFIBUS and PROFINET IO, as specified in the corresponding FDT annexes. The VAN protocol is derived from the PROFINET IO and therefore it must use the same mechanism in a consistent way. The combinations of manufacturer IDs, device type IDs and optionally the module IDs are added to a special location of the Windows registry using a well-defined format. Location and format of the registry keys and values are specified in the TCI specification [TCI1][TCI1.1] and its protocol-dependent annexes [TCI_A1][TCI_A2].

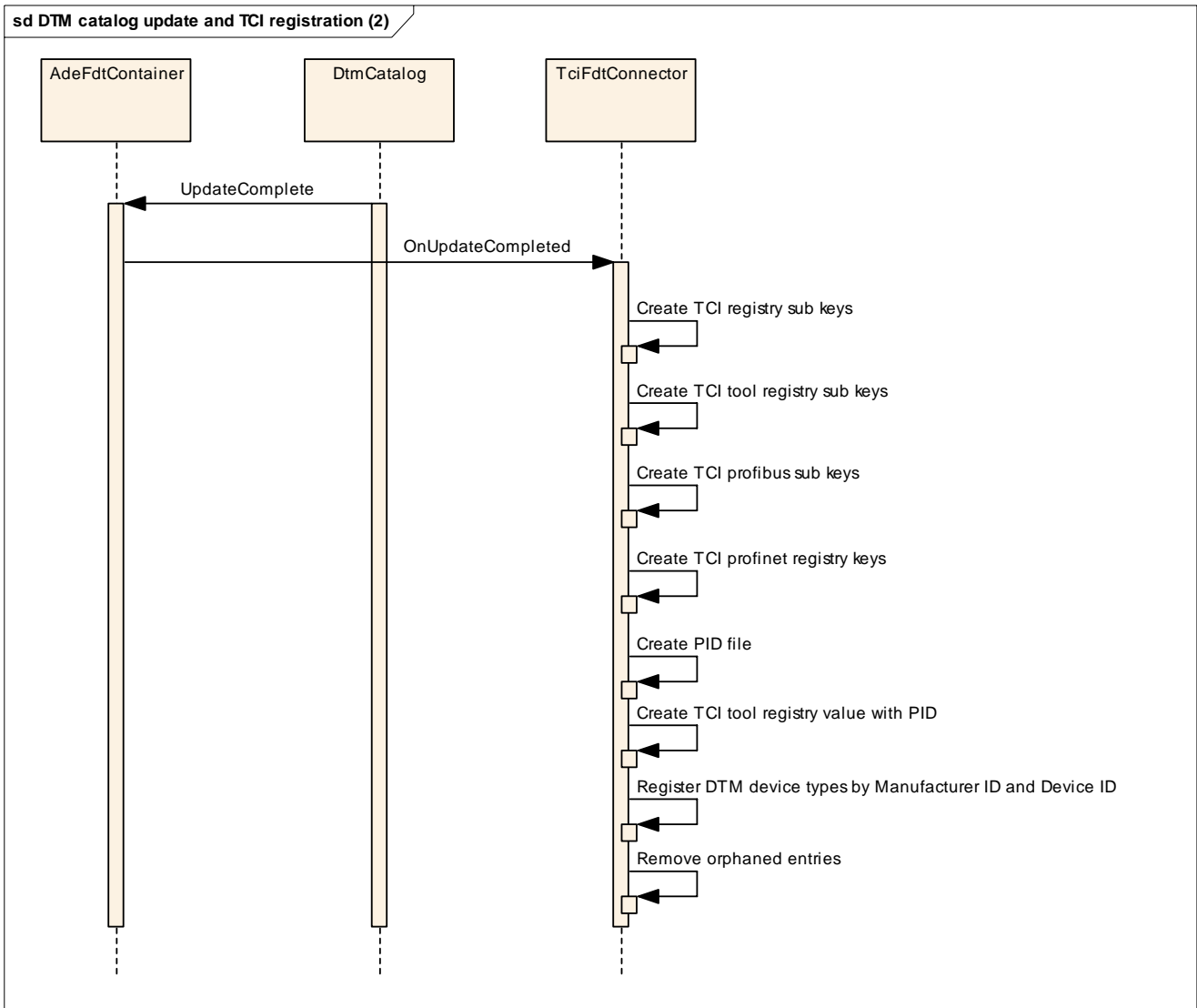


Figure 13: Registration Process part 2

Multiple device tools can be added to TCI section of the Windows registry. Each device that was found in the first stage of the TCI registration process is added to or updated in the Windows registry at the second stage. Each entry points at least to the TCI device tool AutomationXplorer+. An example for that is shown in Figure 14.

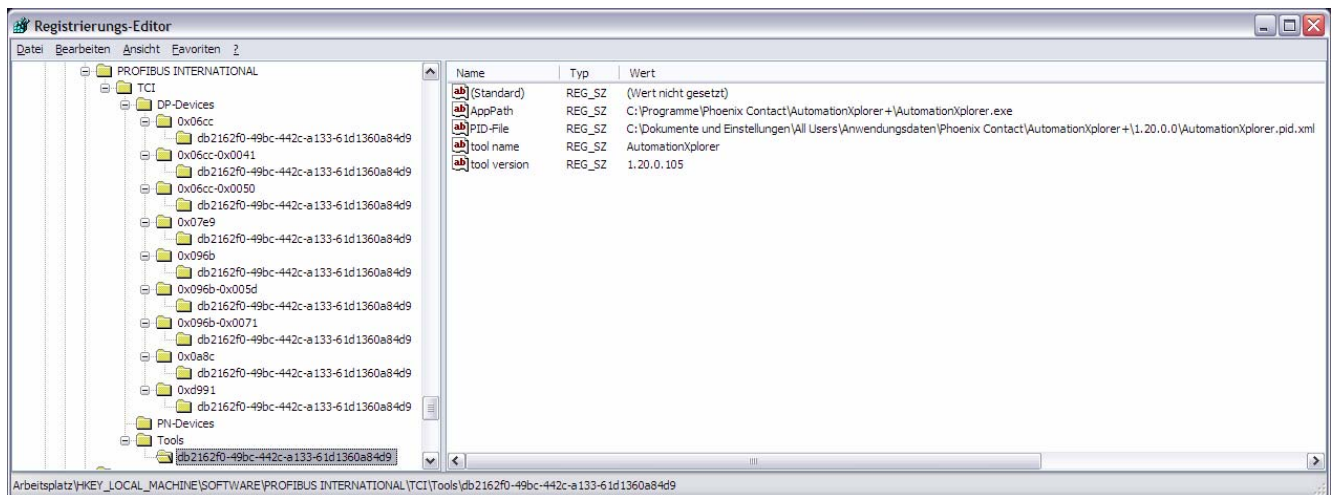


Figure 14: Example for Devices and Tools in the TCI Section of The Windows Registry

5.2 Start-up Phase in TCI Mode

5.2.1 Start-up behaviour of AutomationXplorer+ in TCI Mode

If the AutomationXplorer+ is started in TCI mode, the application runs as singleton. That means it reuses an existing instance of the application that runs in TCI mode.

The AutomationXplorer+ creates a new FDT project, if it is started for the first time with the associated STEP7 project. It loads an existing FDT project, if it was already created and saved at a previous run with the associated STEP7 project. See chapter 5.2.2 concerning to the project information.

In both cases the AutomationXplorer+ closes the current project, if it differs from the new FDT project. The FDT topology is synchronized with the TPF file. The DTMs are added or removed as specified within the TPF file.

5.2.2 Decode TPF File

The TPF file contains three different types of information.

1. The TPF file contains the TCI conformance class of the engineering system STEP7. It is set to the highest conformance level.
2. The TPF file contains project related information. The project related path is provided by STEP7, where a TCI device tool can store its private data. The AutomationXplorer+ loads and stores the FDT project in this folder.
3. The TPF file contains a flat list of all devices that belong to the same subsystem and that are known to the TCI device tool. Each list entry contains type information, the device type ID, and instance related information, the device reference. The device reference is reused as system tag within the AutomationXplorer+. If the AutomationXplorer+ cannot find a device reference also known as system tag from TPF file in its FDT project, a new DTM instance is created. In this case, the DTM type and device type is determined by the manufacturer and device type ID. If one device type is covered by more than one DTM, the AutomationXplorer+ needs the intervention by the user to solve the ambiguity. The device address is updated in the DTM instance by calling `IDtmParameter.GetParameters()` and `IDtmParameter.SetParameters()` as specified in the bus protocol specific FDT annexes [FDT_PB][FDT_PN]. The process is repeated at module and sub module level as well. The AutomationXplorer+ removes DTM instances with system tags whose device reference is not contained in the TPF file.

5.3 Instantiation of DTMs and Transfer of Information

During the synchronization process the AutomationXplorer+ creates new DTM instance for each device item with the device reference that is not known to the FDT project. The AutomationXplorer+ extracts the IdentNumber parameter from the device item. With the IdentNumber the FDT frame application finds a number of corresponding DTMs within the DTM catalogue. The user has to select one DTM in the case that more than one DTM fits to the IdentNumber.

The synchronization process repeats in the same way for the module and the sub module level.

The AutomationXplorer+ starts the synchronization at the level of Communication-DTMs. In contrast to the previously described Device-DTMs the Communication-DTMs are selected by using the bus category from the TPF file. See chapter 6.1.4.

The AutomationXplorer+ copies the data listed in the following table to each DTM instance. The FDT frame application retrieves the DTM parameters from the DTM instances by calling the method `IDtmParameter.GetParameters()`. The returned XML document is updated with the new values. The XML document is written back to the DTM instance by calling the method `IDtmParameter.SetParameters()`.

Table 1: Mapping of Instance Information

Information	Location in TPF file	Location in DTM parameter document
Device address	/InvocationInterface/tpf:DeviceList/ tpf:DeviceItem/tpf:ParameterList/ tpf:ParameterItem[@Name='BusAddress']/ @Value	/FDT/DtmDevice/BusInformation/ UserDefinedBus/DeviceList/ pnio:Network/@ipAddress
FDT tag	/InvocationInterface/tpf:DeviceList/ tpf:DeviceItem/@Name	/FDT/DtmDevice/@fdt:tag
VAN instance model	/InvocationInterface/tpf:DeviceList/ tpf:DeviceItem/tpf:ParameterList/ tpf:ParameterItem[@Name='TciExtraData']/ @Value	/FDT/DtmDevice/ ExportedVariables/ fdt:DtmVariables/ fdt:DtmVariable [@nodeId='TciExtraData']/ fdt:Value/fdt:Variant/ fdt:StringData/@string

5.4 Implementation of FDT Interfaces

The AutomationXplorer+ implements the whole functionality that is listed below in a single process. Because the application is realized with the Microsoft .NET Framework Version 2.0, DTMs can be realized with .NET Framework 2.0 or .NET Framework 1.1 as well.

Level of implementation for each interface function defined in FDT specification that has to be implemented on the FDT frame side:

Table 2: Description of Implementation Levels

Level of implementation	Description
Full	The function provides a full implementation.

Basic	The function contains a simple implementation at least a mapping to events that can be handled by additional plug-ins.
Dummy	The function returns immediately. This is required for mandatory interfaces.
Not supported	The application does not provide the corresponding interface. This is possible only for optional interfaces.

Table 3: Implementation Levels for the Frame Application Interfaces

Frame Application Interfaces	Operations	Level of implementation
IDtmEvents	OnApplicationClosed	Full
	OnDownloadFinished	Full
	OnErrorMessage	Full
	OnFunctionChanged	Full
	OnChannelFunction-Changed	Full
	OnInvokedFunction-Finished	Full
	OnNavigation	Dummy
	OnOnlineStateChanged	Full
	OnParameterChanged	Full
	OnPreparedToRelease	Full
	OnPreparedToRelease-Communication	Full
	OnPrint	Dummy
	OnProgress	Full
	OnScanResponse	Dummy
OnUploadFinished	Full	
IDtmEvents2	OnStateChanged	Full
IDtmScanEvents	OnScanReponse	Full
	OnScanHardware-Response	Full
IDtmAuditTrailEvents	OnAuditTrailEvent	Basic
	OnEndTransaction	Basic
	OnStartTransaction	Basic
IFdtActiveX	CloseActiveXControl-Request	Full
	OpenActiveXControl-Request	Full
IFdtActiveX2	OpenDialogActiveX-ControlRequest	Full

	OpenDialogChannel-ActiveXControlRequest	Full
	CloseChannelActiveX-ControlRequest	Full
	OpenChannelActiveX-ControlRequest	Full
IFdtBulkData	GetInstanceRelatedPath	Not Supported
	GetProjectRelatedPath	Not Supported
IFdtContainer	GetXmlSchemaPath	Full
	LockDataSet	Basic
	SaveRequest	Full
	UnlockDataSet	Basic
IFdtDialog	UserDialog	Basic
IFdtTopology	CreateChild	Dummy
	DeleteChild	Dummy
	GetChildNodes	Full
	GetDtmForSystemTag	Full
	GetDtmInfoList	Full
	GetParentNodes	Full
	MoveChild	Dummy
	ReleaseDtmForSystem-Tag	Full
IDtmRedundancyEvents	OnAddedRedundantChild	Not Supported
	OnRemovedRedundant-Child	Not Supported
IDtmSingleDeviceData-AccessEvents	OnItemListResponse	Basic
	OnDeviceItemListChange d	Basic
	OnReadResponse	Basic
	OnWriteResponse	Basic
IDtmSingleInstanceData-AccessEvents	OnInstanceItemList-Changed	Basic
IFdtBtmTopology	CreateChild	Not Supported
	DeleteChild	Not Supported
	GetChildNodes	Not Supported
	GetBtmForSystemTag	Not Supported
	GetBtmInfoList	Not Supported
	GetParentNodes	Not Supported
	MoveChild	Not Supported

	ReleaseBtmForSystem-Tag	Not Supported
--	-------------------------	---------------

6 VAN Communication DTM

6.1 Architecture of VAN Communication DTM

In this section the architecture of the *VAN Communication DTM* is presented. Firstly, a general overview is presented to analyse the way the *VAN Communication DTM* interacts with other components. Secondly, an inside view is shown with more detail about the internal structure of the *VAN Communication DTM*, including a class view over the *VAN Communication DTM*. Lastly, the data flow between this component, the *VAN Device DTM* and the *VAN device* is explained.

6.1.1 General Overview of VAN Communication DTM

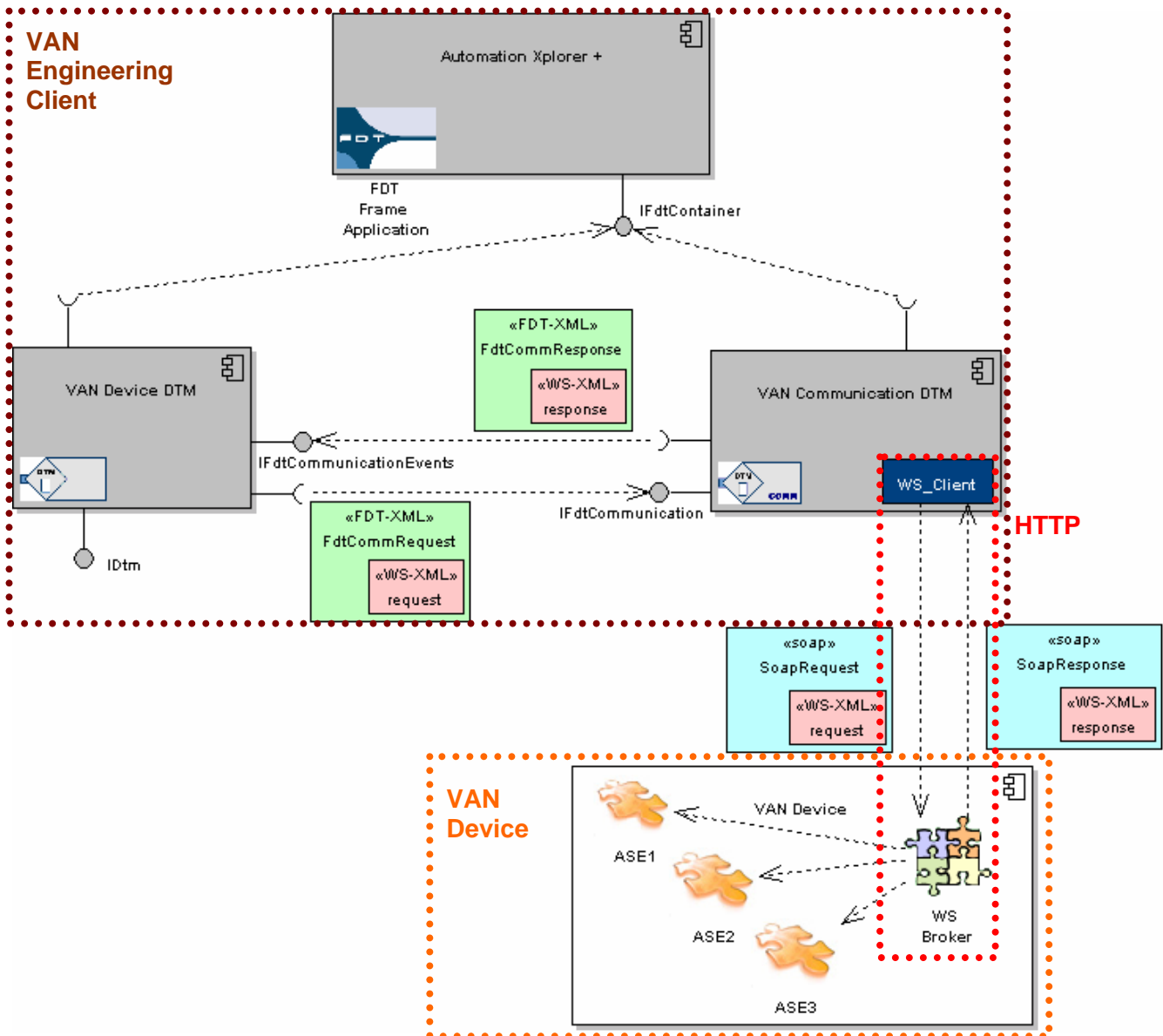


Figure 15: General Overview of the VAN Communication DTM

The component model shown in Figure 15: visualises the main interactions between the different modules. The component model contains four modules, *AutomationXplorer+*, *VAN Communication DTM*, *VAN Device DTM* and the *VAN device*. All these components exchange information and it is possible to observe the XML documents resulting from these interactions in the diagram.

As there are different documents with related content in the diagrams, the following terminology is used for unique identification of each document:

FDT-XML:

FdtCommRequest

This document contains the necessary information to make a connection, a request and commands between a *VAN Device DTM* and a *VAN Communication DTM*. The *FdtCommRequest* document includes the WS-XML request document.

FdtCommResponse

This document contains the necessary information to receive a response to a *VAN Device DTM* from a *VAN Communication DTM*. The *FdtCommResponse* document includes the WS-XML response document.

SOAP:

SoapRequest

This is the Soap request used to send an ASE from the *WS Client* to the *VAN device*. The *SoapRequest* document includes the WS-XML request document.

SoapResponse

This is the Soap response to the request of the *WS Client*. The *SoapResponse* document includes the WS-XML response document.

WS-XML:

request

This document contains the information about one ASE and the data necessary to perform an operation of the Web Service. It represents a request for an operation in the *VAN device*.

response

This document contains the information about one ASE and the returned data from the operation of the Web Service. It represents a response to an operation performed in the *VAN device*.

6.1.2 Internal Overview of VAN Communication DTM

The class diagram shown in Figure 16 depicts in detail all the classes that constitute a *VAN Communication DTM*. Some classes implement FDT interfaces. More information about the FDT interfaces is given in chapter 6.3.

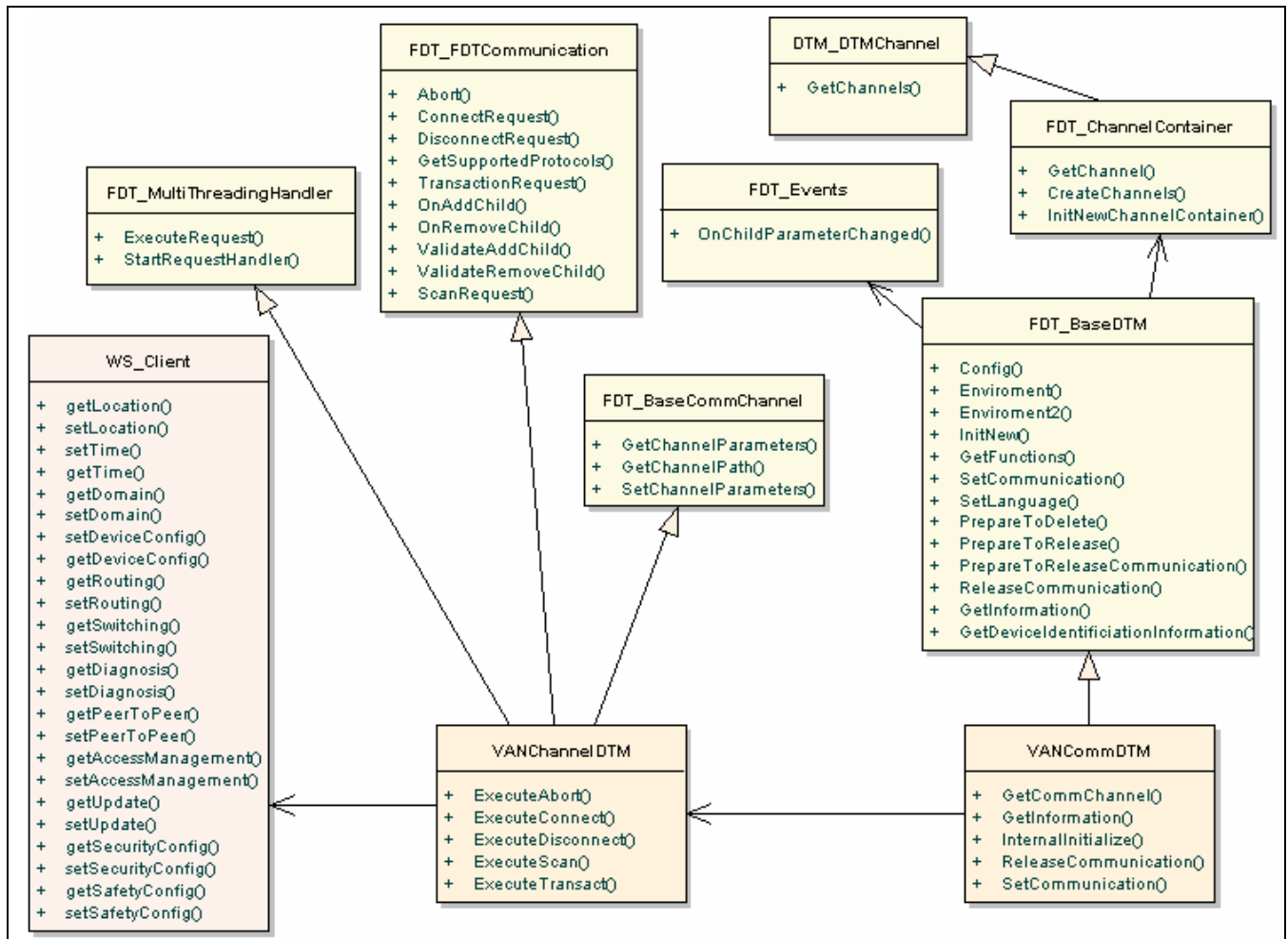


Figure 16: Class Diagram of VAN Communication DTM

Two main classes are shown at the bottom of the diagram: *VANChannelDTM* and *VANCommDTM*. These classes inherit or include other classes. The descriptions of the classes as well as the implemented interfaces are presented in the following.

VANCommDTM

Description:

Represents the communication from the *VAN Communication DTM* to the *FDT Frame Application*. This class inherits the functionalities from the *FDT_BaseDTM*.

VANChannelDTM

Description:

Encapsulates the communication part of the *VAN Communication DTM*. This class inherits the functionalities of the classes *FDT_MultiThreadingHandler*, *FDT_FDTCommunication* and *FDT_BaseCommChannel*.

*FDT_MultiThreadingHandler***Description:**

This class has the responsibility for the generation of threads of execution, used for receiving the messages, which arrive with the FDT interface IFdtCommunication.

*FDT_FDTCCommunication***Description:**

This class is used to implement the communication server for a channel object for the communication with the *VAN Device DTM*. This class also implements the functionality to retrieve topology information.

Implemented interfaces:

IFdtCommunication
IFdtChannelSubTopology

*FDT_BaseCommChannel***Description:**

This class implements the functionality of the FDT interface IFdtChannel. It represents the base for each communication channel object.

Implemented interfaces:

IFdtChannel

*FDT_BaseDTM***Description:**

This class contains the functions of the FDT interfaces IDtm, IDtm2 and IDtmInformation. Furthermore, it has functions to handle XML.

Implemented interfaces:

IDtm
IDtm2
IDtmInformation

*FDT_Events***Description:**

This class handles the FDT events coming to a DTM.

Implemented interfaces:

IFdtEvents

*FDT_ChannelContainer***Description:**

This class is a DTM channel container which is used to store several DTMchannel.

*DTM_DTMChannel***Description:**

This class contains the information about a DTMchannel.

Implemented interfaces:

IDtmChannel

*WS_Client***Description:**

This class is responsible for the communication with the *VAN device*, following the Web Services architecture.

6.1.3 VAN Communication DTM Interactions

The interaction between *VAN Communication DTM*, *VAN Device DTM* and *VAN device* is shown in Figure 17. The communication between components follows the *VAN communication schema*. It is important to highlight the documents, which are exchanged between components. To know more about these documents please refer to chapter 6.1.1. The actual interaction represents an example to set a VAN Domain ASE.

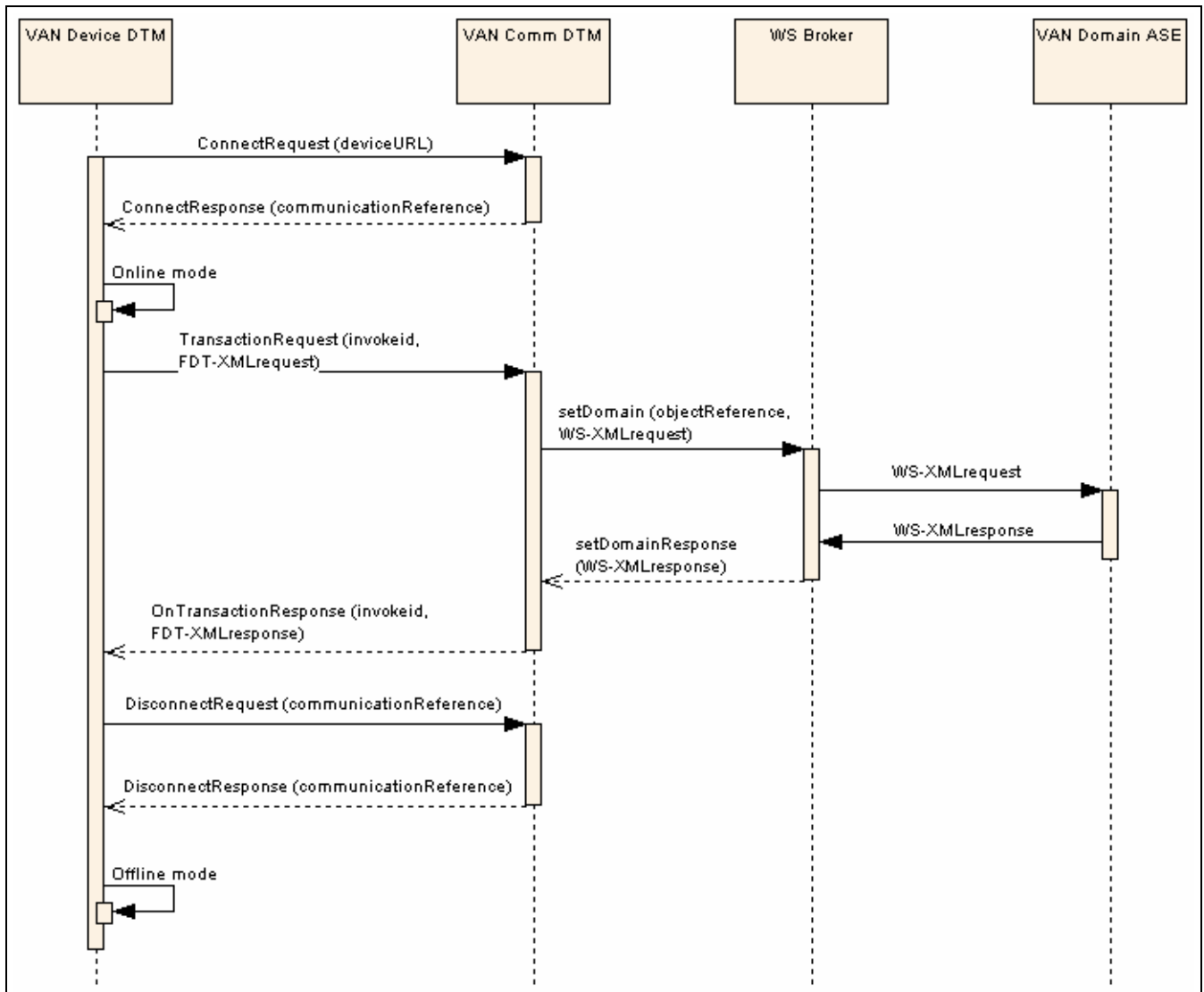


Figure 17: Behaviour Diagram of VAN Communication DTM

6.1.4 Bus Category Identifier

The bus category identifier is used to announce a new communication protocol in FDT. With the bus category identifier a FDT Frame Application can identify if a *VAN Device DTM* fits to a *VAN Communication DTM*. This bus category identifier is a Universal Unique Identifier (UUID). The UUID is nothing more than a 128-bit number.

The following number is defined as the bus category identifier of the communication protocol for the VAN project:

{294CCA43-6AD2-4736-A149-710BFDEB5147}

6.2 VAN Communication Schema

The purpose of this chapter is to present a solution to generate *VAN Communication Schemas*, and upgrade some information about this schema. A short introduction about this schema is given first, and then an algorithm is presented, which is used to derive this schema from existing documents. After this, it is explained the structure and features of a tool generator to derive this schema. Finally, a diagram to exemplify a communication with this schema is presented.

The *VAN Communication Schema* was presented in deliverable [D08.4-2]. This schema is used to validate XML messages sent between the *VAN Device DTM* and the *VAN Communication DTM*. As explained in deliverables [D08.4-1] and [D08.4-2] the *VAN Communication Schema* must define the operations specific for the FDT communication and the operations specific for VAN.

The VAN specific operations have a one-to-one correspondence to the *WSDL schema*. Each time that a new ASE is created, changed or eliminated, the *VAN Communication Schema* needs to be changed as well in order to adapt the *VAN Device DTM–VAN Communication DTM* communication protocol accordingly. In addition, the WSDL has to be changed to be compliant to the new ASE. In order to get the task of generating a new *VAN Communication Schema* easier and to avoid vulnerability by human mistakes a tool is developed to generate a new *VAN Communication Schema* from the new WSDL version. The tool used for this task is called *VAN Communication Schema Generator*.

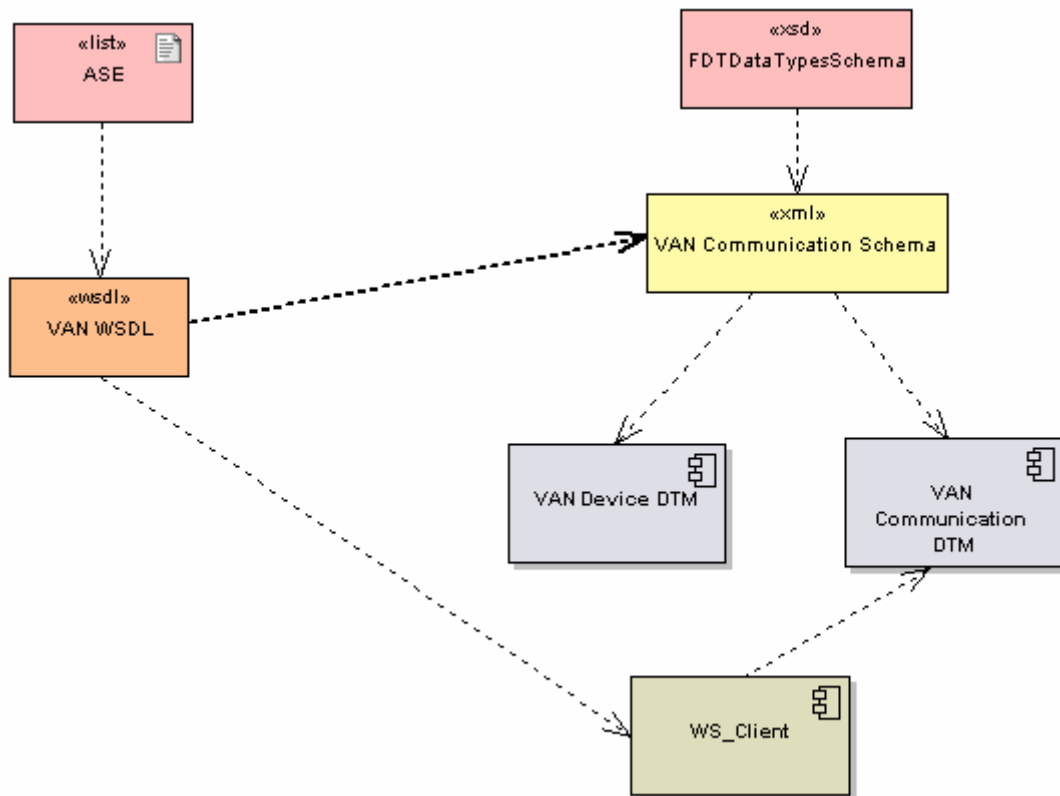


Figure 18: Dependencies of Documents and Components

The schema in Figure 18 represents the dependencies between the several documents in the VAN project. The definition of the WSDL document is depending on the definition of the ASEs. The *WS Client* as well as the *VAN Communication Schema* depends on the WSDL document definition. Moreover, the two DTM components are dependent on the *VAN Communication Schema*. The dependency between *VAN Communication Schema* and *VAN WSDL* is represented with a dashed arrow in bold.

6.2.1 Schema Transformation Algorithm

The way to generate new ASE, which is described by WSDL, to the *VAN Communication Schema* can be automated. The WSDL *portType* tag gives the information about the ASE operations. Each operation has an input and output, correspondent to Request or Response. With this information the attributes to be derived are known. For a Request there are three attributes, *communicationReference*, *objectReference* and *xmlRequest*. For the response there are two attributes, *communicationReference* and *xmlResponse*.

The following XML is a part of the original WSDL document. This XML document presents just two of the several operations of the WSDL, *getDomain* and *setDomain*.

WSDL – VAN_services

```
<.....>
<wsdl:portType name="VAN_Services">
  <wsdl:operation name="getDomain">
    <wsdl:input name="getDomainRequest" message="impl:getDomainRequest"/>
    <wsdl:output name="getDomainResponse" message="impl:getDomainResponse"/>
  </wsdl:operation>
  <wsdl:operation name="setDomain">
    <wsdl:input name="setDomainRequest" message="impl:setDomainRequest"/>
    <wsdl:output name="setDomainResponse" message="impl:setDomainResponse"/>
  </wsdl:operation>
</wsdl:portType>
<.....>
```

The schema that follows is a newer version of the *VAN Communication Schema*, which was presented in deliverable [D08.4-2]. The differences from the version in D08.4-2 are highlighted. The main differences are the names of the operations. In this version, there are only operations available for the *VAN Domain ASE*. These operations were derived / generated from the WSDL above. Four functions were derived, *setDomainRequest*, *setDomainResponse*, *getDomainRequest* and *getDomainResponse*. Moreover, a header is now added by the *VAN Communication Schema Generator Tool*, which will be specified in the next chapters. The header contains the basic information about the tool, the author and the date.

VAN Communication Schema

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

```
<!--
```

This XML document was automatically generated by *VAN Communication Schema Generator 1.0*
From WSDL version: 1.1

Author name: <windows login name>

Date: <system date when the file was generated>

```
-->
```

```
<Schema name="VANCommunicationSchema"
  xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schemas-microsoft-com:datatypes"
  xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <AttributeType name="schemaVersion" dt:type="number" default="0.1"/>
  <AttributeType name="communicationReference" dt:type="uuid"/>
  <AttributeType name="objectReference" dt:type="string"/>
  <AttributeType name="xmlRequest" dt:type="string"/>
  <AttributeType name="xmlResponse" dt:type="string"/>
  <AttributeType name="deviceURL" dt:type="string"/>
```

```

<!-- VAN DTM Transactions -->
<!-- common for FDT -->
<ElementType name="ConnectRequest" content="eltOnly" model="closed">
  <attribute type="deviceURL" required="yes"/>
</ElementType>
<ElementType name="ConnectResponse" content="empty" model="closed">
  <attribute type="communicationReference" required="yes"/>
</ElementType>
<ElementType name="DisconnectRequest" content="empty" model="closed">
  <attribute type="communicationReference" required="yes"/>
</ElementType>
<ElementType name="DisconnectResponse" content="empty" model="closed">
  <attribute type="communicationReference" required="yes"/>
</ElementType>
<!-- special for VAN -->
<ElementType name="setDomainRequest" content="empty" model="closed">
  <attribute type="communicationReference" required="yes"/>
  <attribute type="objectReference" required="no"/>
  <attribute type="xmlRequest" required="no"/>
</ElementType>
<ElementType name="setDomainResponse" content="empty" model="closed">
  <attribute type="communicationReference" required="yes"/>
  <attribute type="xmlResponse" required="no"/>
</ElementType>
<ElementType name="getDomainRequest" content="empty" model="closed">
  <attribute type="communicationReference" required="yes"/>
  <attribute type="objectReference" required="no"/>
  <attribute type="xmlRequest" required="no"/>
</ElementType>
<ElementType name="getDomainResponse" content="empty" model="closed">
  <attribute type="communicationReference" required="yes"/>
  <attribute type="xmlResponse" required="no"/>
</ElementType>
<!-- Main FDT element -->
<ElementType name="FDT" content="eltOnly" order="one" model="closed">
  <attribute type="schemaVersion"/>
  <attribute type="fdt:nodeId"/>
  <group order="one" maxOccurs="1" minOccurs="1">
    <element type="ConnectRequest"/>
    <element type="ConnectResponse"/>
    <element type="DisconnectRequest"/>
    <element type="DisconnectResponse"/>
    <element type="setDomainRequest"/>
    <element type="setDomainResponse"/>
    <element type="getDomainRequest"/>
    <element type="getDomainResponse"/>
    <element type="fdt:CommunicationError"/>
  </group>
</ElementType>
</Schema>

```

6.2.2 VAN Communication Schema Generator

This section describes the tool to generate *VAN Communication Schemas*. As explained above this tool generates *VAN Communication Schemas* from WSDL documents. This tool is called *VAN Communication Schema Generator*, *VAN CSGen* for short. The next sections present the architecture, the user interface and the general behaviour of this tool.

The *VAN CSGen* has a three-tier architecture as shown in Figure 19. With this architecture it is possible to achieve easy-to-replace module logic. So if the WSDL version changes from 1.1 to 1.2 for example, it is easy to substitute or to expand the *Data Tier* responsible for the XML document manipulation. This architecture provides an easy way to replace the Presentation Tier, for the case that more functionality is needed without having to be worried about XML manipulation.

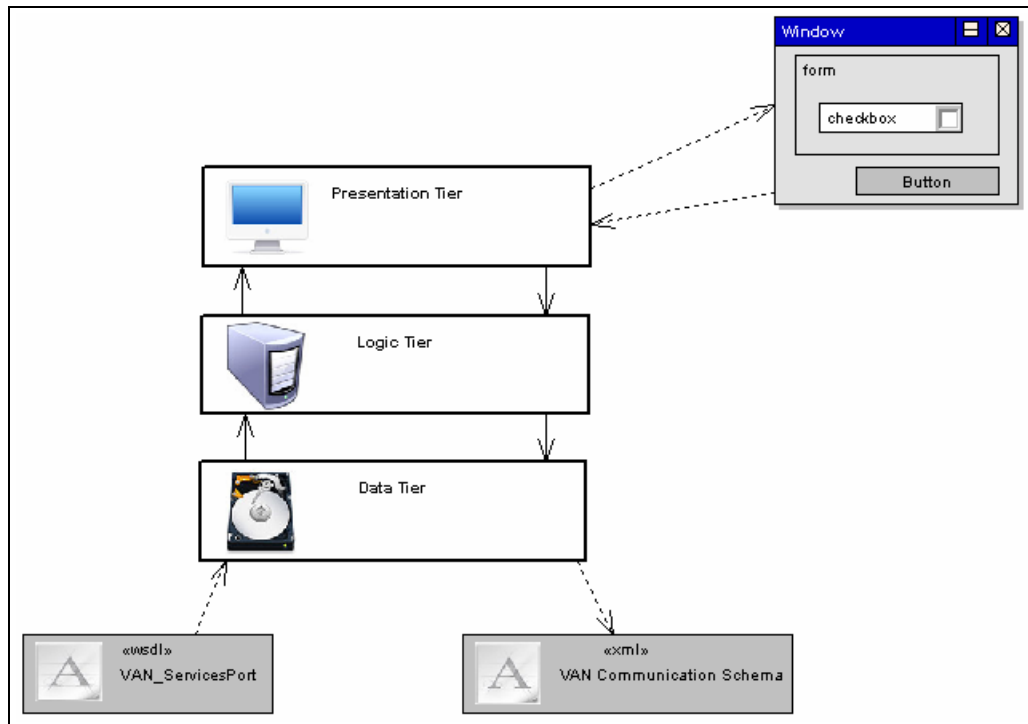


Figure 19: VAN CSGen Architecture

The internal architecture of the tool is presented in Figure 20. The class *GUI* belongs to the *Presentation Tier*. The *BusinessLogic* class belongs to the *Logic Tier*. And the rest of the classes, *CommSchemaGenerator*, *WSDLParser*, *SchemaLocation* all belong to the *Data Tier*.

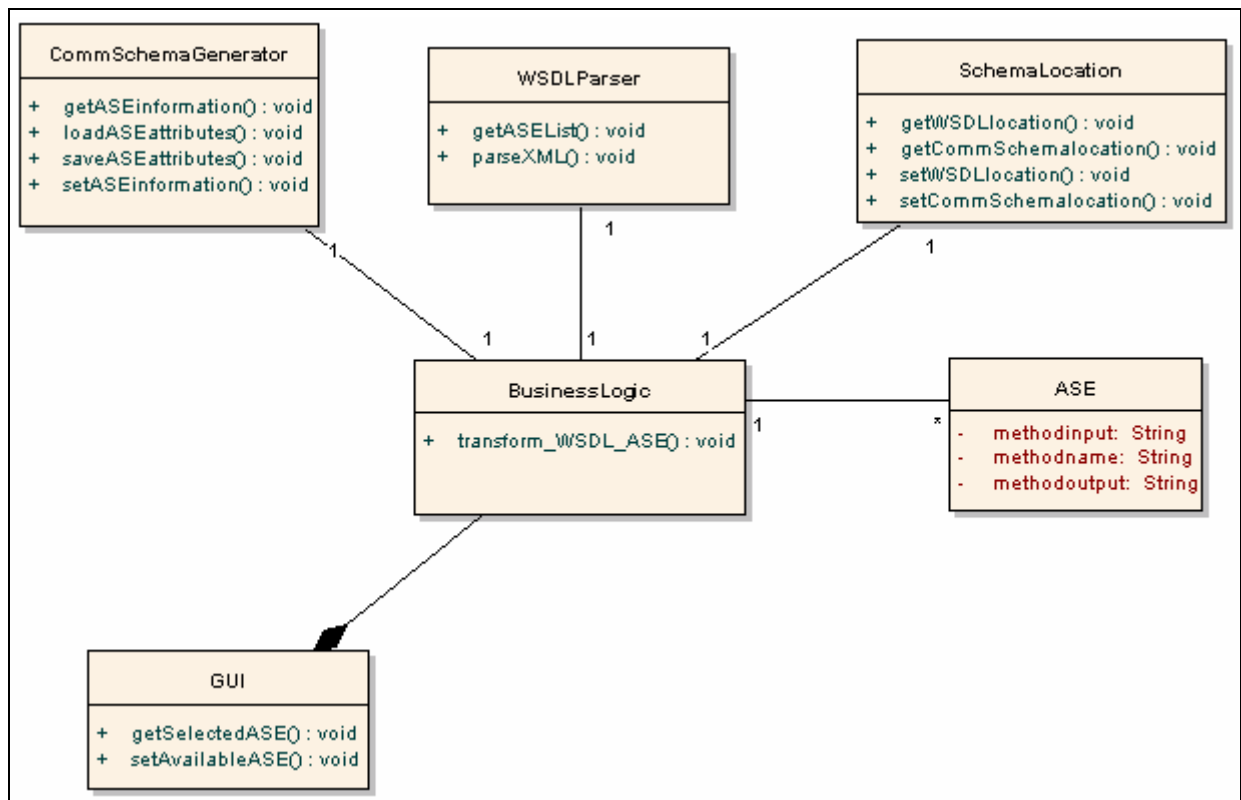


Figure 20: VAN Communication Schema Generator Class Diagram

The individual classes of the different tiers are described in the following:

Presentation Tier:

GUI:

This class is responsible for presenting the information. It has two important methods to be used by the *BusinessLogic* class: *getSelectedASE()* to read the ASEs selected by the user and *setAvailableASE()* to generate the options to be used by the user.

Logic Tier:

BusinessLogic:

This class is responsible for the orchestration of the program, being responsible for the exceptions handling, and behaving as a layer between presentation and data manipulation.

Data Tier:

CommSchemaGenerator

This class is responsible to parse the template of the *VAN Communication Schema*, to find the nodes where the information about the ASEs have to be provided and to generate the VAN Communication Schema correctly.

WSDLParser

This class is responsible to parse the WSDL. The main function of the class is *getASEList()*, to get the ASEs available in the WSDL.

SchemaLocation

This class is responsible to get the Windows registry attributes concerning the location of the files for the schemas. Furthermore, it is responsible to change these locations.

ASE class:

This class is the main data structure used and it represents an ASE. This class has the following attributes concerning the information needed for the *VAN Communication Schema* generation: *methodname* for the name of the method, *methodinput* and *methodoutput* for the corresponding *wsdl:input* and *wsdl:output* operation names.

The use of the Windows registry is a useful feature, making the schema generation an almost one-click task. In the Windows registry two important entries are saved: a string with the *WSDL Schema Location*, and another string with the *VAN Communication Schema Template Location*. For default it uses a subfolder of the installation folder for the tool (C:\\Program Files\\VAN Project\\VAN Communication Schema Generator\\xml). The *VAN Communication Schema Template* is used, so the user does not have to provide a version of the document. Additionally, the default path to be used for saving the generated VAN Communication Schema is also stored in the Windows registry.

The Windows Registry keys are specified in the next lines:

Windows Registry Editor Version 5.00

```
[HKEY_LOCAL_MACHINE\ SOFTWARE\VAN\CSGEN]
```

```
"Softwarepath"="C:\\Program Files\\VAN Project\\VAN Communication Schema Generator"
```

```
[HKEY_LOCAL_MACHINE\ SOFTWARE\VAN\CSGEN]
```

```
"WSDLSchema"="C:\\Program Files\\VAN Project\\VAN Communication Schema Generator\\xml\\  
VAN_ServicesPort.wsdl"
```

```
[HKEY_LOCAL_MACHINE\ SOFTWARE\VAN\CSGEN]
```

```
"CommSchemaTemplate"="C:\\Program Files\\VAN Project\\VAN Communication Schema  
generator\\xml\\ VAN_CommTemplate.xml"
```

```
[HKEY_LOCAL_MACHINE\ SOFTWARE\VAN\CSGEN]
```

```
"CommSchema"="C:\\Program Files\\VAN Project\\VAN Communication Schema generator\\result\\  
VAN_CommSchema.xml"
```

VAN CSGlobal - User Interface Description

The user interface of the tool is shown in Figure 21 along with the different options and areas of the program. It consists of one window making the process of creating new VAN Communication Schemas simple and quick.

VAN CSGlobal User Interface - Main Window

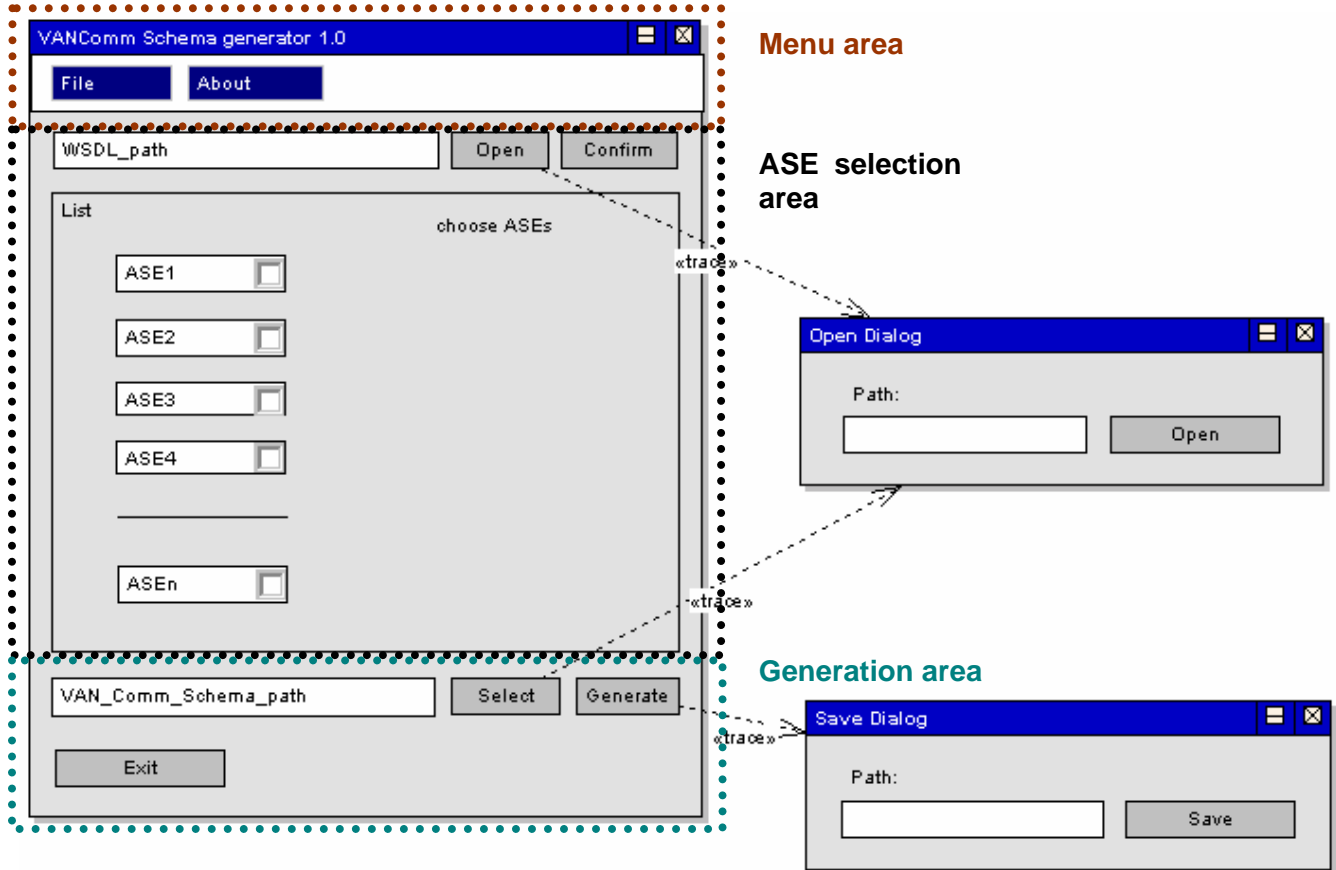


Figure 21: VAN CSGlobal Main Window

The main window is divided in three areas:

Menu area

In this area it is possible to access the configurations, open files, change locations, access about information. A detailed description of the different menu options is given in the next section.

ASE selection area

This area is generated when the WSDL path is confirmed. Checkboxes are generated for each ASE derived from the WSDL. Depending on the selection state of the checkboxes the operations for the corresponding ASEs are included in the generated VAN Communication Schema or not.

Generation area

In this area is possible to choose an alternative location to the generated VAN Communication Schema and to start generation of the final document.

VAN CSGlobal User Interface - Menu Options

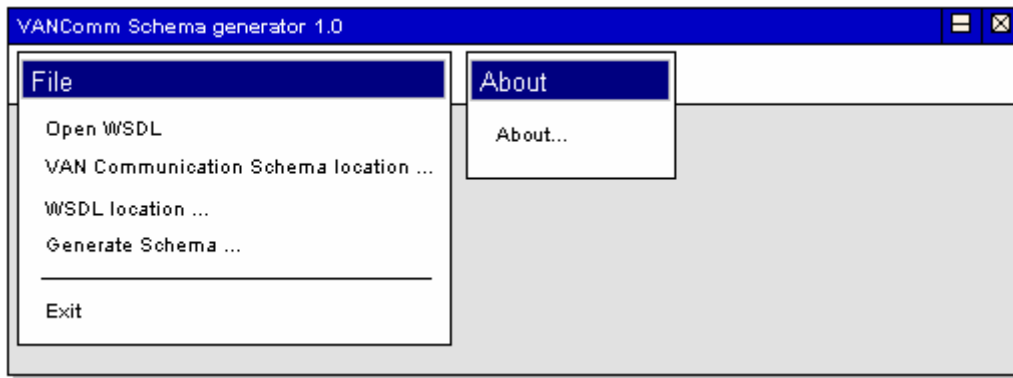


Figure 22: VAN CSGen Menu Options

Figure 22 shows the different menu options that can be chosen before generation.

File

Open WSDL

This option opens a WSDL.

VAN Communication Schema location...

With this option it is possible to change the VAN Communication Schema Template location.

WSDL location ...

This option allows the user to change the default WSDL location.

Generate Schema ...

This option generates the VAN Communication Schema.

Exit

This option makes the program to quit.

About

About...

In this option it is possible to access the version, the author, and the company providing the tool.

VAN CSGen - How to Use the Tool

When the tool is launched a window like the one shown in Figure 21 appears to the user. As described before this window has three areas that have to be used one after the other.

1. The first step is to define a WSDL location. This WSDL must be provided in order for the tool to get the list of ASEs. By default, it is a location of a tool subfolder. This location can be changed in File→Change WSDL location...
2. After choosing and confirming a WSDL location, the ASEs available in this document generate a list of checkboxes. The user selects or deselects the ASEs.
3. Finally, the user can generate the VAN Communication Schema, by pressing the "Generate" button. The VAN Communication Schema is saved in the default location. This location can be changed before the generation by opening a location.

VAN CSGen - Components Behaviour

The interactions between the main classes of the tool are specified in Figure 23. In the diagram the two classes *WSDLParser* and *CommSchemaGenerator* are explicitly stated. The other classes are represented by the mean of *VAN CSGen Tool* to simplify the diagram. The specification for the different classes is given in the beginning of chapter 6.2.2

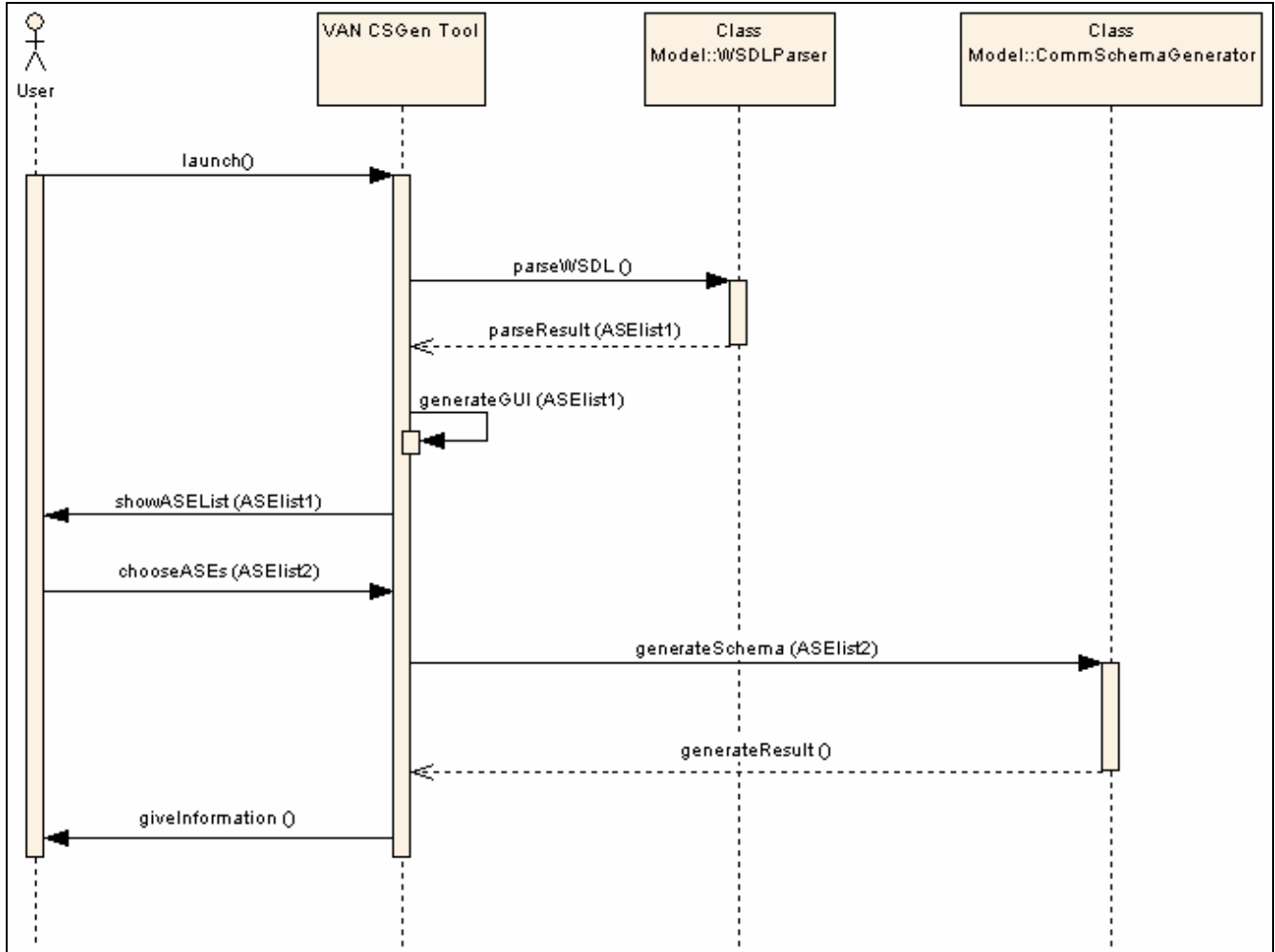


Figure 23: VAN CSGen Behaviour

6.3 Implementation of FDT/DTM Interfaces

This section presents an overview about the FDT interfaces, which will be implemented in the prototype for the *VAN Communication DTM*. For the prototype, not all the interfaces defined in the FDT specification [FDT1.2.1] need to be implemented fully. Some interfaces have to be implemented fully while others can have just a dummy implementation. All the interfaces and their functions can be found in [FDT1.2.1] if more information is needed.

In Figure 24 it is possible to identify who is using which interface from the *VAN Communication DTM*. Only the interfaces that have all the methods fully implemented are shown in this figure.

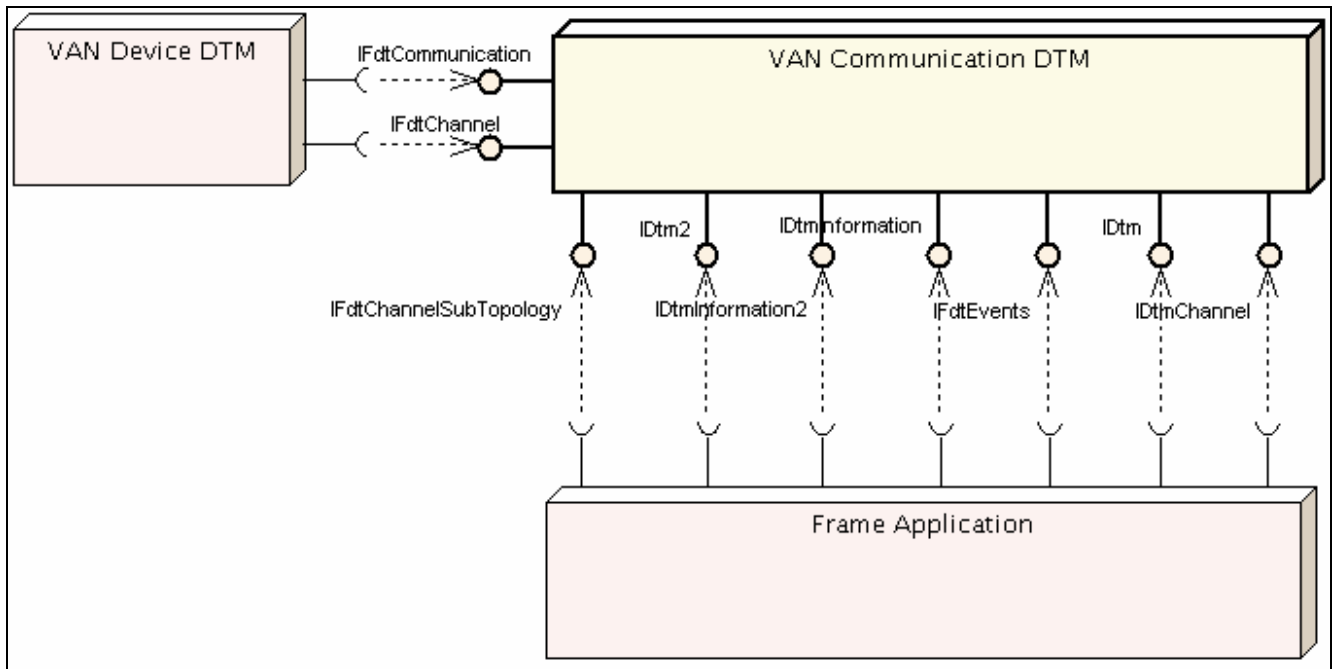


Figure 24: VAN Communication DTM Interfaces Usage

In Figure 25 it is possible to have a better view of the interfaces to be implemented in the prototype of the *VAN Communication DTM*. Each interface is coloured depending on the level of the implementation for the prototype. For a dummy implementation, the interface is coloured black and levels of gray until *beige* for an interface fully implemented. These levels depend on the number of functions being implemented.

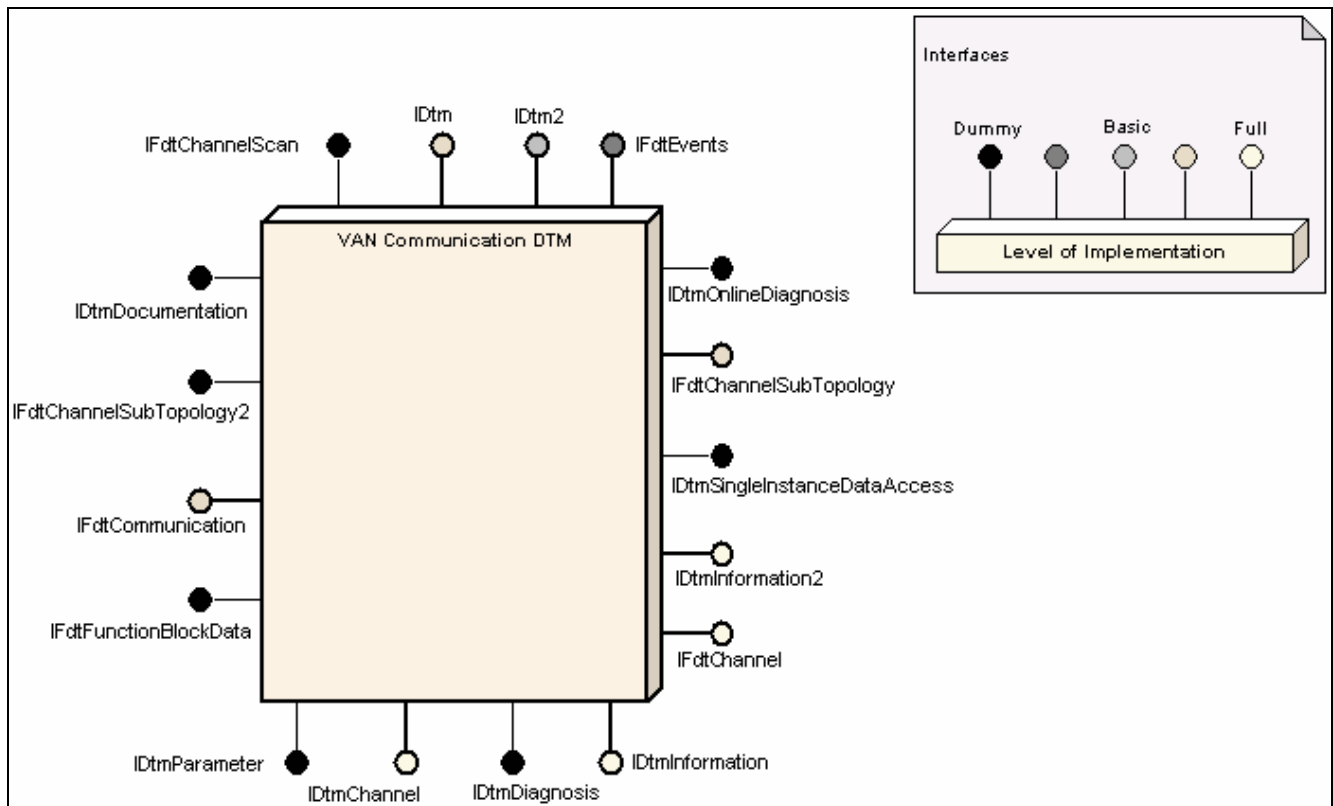


Figure 25: VAN Communication DTM Interfaces - Level of Implementation

Table 4: VAN Communication DTM Interfaces

Interfaces	Functions	Level of implementation
IDtm	Config	Full
	Environment	Full
	GetFunctions	Full
	InitNew	Full
	Invoke Function request	Dummy Ignoring all input parameters and return FALSE. No function is provided by the VAN Communication DTM
	PrepareToDelete	Full
	PrepareToRelease	Full
	PrepareToReleaseCommunication	Full
	PrivateDialogEnabled	Dummy Ignoring input parameter. Independent of the input parameter the method returns TRUE. VAN

		Communication DTM will never open private dialogs
	ReleaseCommunication	Full
	setCommunication	Full
	SetLanguage	Full
IDtm2	Environment2	Full
	SetSystemGuiLabel	Dummy Ignoring input parameter. Independent of the input parameter the method returns TRUE. VAN Communication DTM will never open private dialogs
IDtmChannel	GetChannels	Full
IDtmDocumentation	GetDocumentation	Dummy This method returns an XML document containing the empty documentation specified by the DTMDocumentationSchema (this schema is described in the FDT Specification document)
IDtmDiagnosis	Compare	Dummy This method will not work properly for the prototype. It will always return FALSE meaning that the comparison failed
	InitCompare	Dummy As the Compare() method is not implemented in the prototype, this method returns FALSE, so the method Compare() can not be called
	ReleaseCompare	Dummy This method returns TRUE, meaning "Compare sequence closed"
	Verify	Dummy This method is used to verify if the data is consistent. This method returns TRUE
IDtmInformation	GetInformation	Full
IDtmInformation2	GetDeviceIdentificationInformation	Full
IDtmOnlineDiagnosis	Compare	Dummy This method returns an XML file with 'noComparableData' as value of the attribute 'statusFlag'
	GetDeviceStatus	Dummy This method returns a dummy XML

		file document
IDtmParameter	GetParameters	Dummy This method returns a XML document conformant to the DTMPParameterSchema, but with no parameters. The parameters of the VAN Communication DTM are not public
	setParameters	Dummy No parameters are settable
IFdtEvents	OnChildParameterChanged	Full
	OnLockDataSet	Dummy This method has no return value. It is used in a multi-user situation as a signal to the DTMs when someone wants to access critical data
	OnParameterChanged	Dummy
	OnUnlockDataSet	Dummy The prototype does not consider the multi-user possibility
IDtmSingleInstanceData Access	GetItemList	Dummy This method returns a XML document containing an empty DtmItemInfoList
	Read	Dummy This method returns an empty document
	Write	Dummy This method returns an empty document
IFdtChannel	GetChannelParameters	Full
	GetChannelPath	Full
	SetChannelParameters	Full
IFdtChannelSubTopology	OnAddChild	Full
	OnRemoveChild	Full
	ScanRequest	Dummy This method returns FALSE meaning that the ScanRequest operation failed
	ValidateAddChild	Full
	ValidateRemoveChild	Full
IFdtChannelSubTopology2	SetChildrenAddresses	Dummy This method returns a XML document conformant with the

		DTMDeviceListSchema and the field DeviceList with no attributes
IFdtCommunication	Abort	Full
	ConnectRequest	Full
	DisconnectRequest	Full
	GetSupportedProtocols	Full
	SequenceBegin	Dummy This method returns FALSE. The prototype does not support communication sequences
	SequenceEnd	Dummy This method returns FALSE. The prototype does not support communication sequences
	SequenceStart	Dummy This method returns FALSE meaning that "Communication sequence could not be started". The prototype does not support communication sequences
	TransactionRequest	Full
IFdtFunctionBlockData	GetFBInstanceData	Not implemented This method could be implemented to do the propagation of the failsafe function calls, but for prototyping failsafe is not considered
	SelectFBInstance	Not implemented This method could be implemented to do the propagation of the failsafe function calls, but for prototyping failsafe is not considered
IFdtChannelScan	ScanRequest	Dummy This method returns FALSE. This interface is only used in the version 1.2.1 of FDT Specification. Otherwise, the same methods in the IFdtChannelSubTopology interface are used
	CancelAction	Dummy This method returns FALSE, meaning that the channel cannot cancel the operation

6.4 Implementation of Web Service Client

The Web Service client is implemented by the reusable component *WS_Client* (see also [D08.5-1]). The implementation of the *WS_Client* component can be a very automatic task, once the WSDL is available. The *WS_Client* is also very easy to update when it only depends on the WSDL. Whenever another ASE is defined the WSDL has to be changed, and from the WSDL it is easy to derive another *WS_Client*.

In the market, very good frameworks for the generation of Web Service clients are available, as examples, gSOAP, .NET Framework and Axis, just to mention the most popular ones.

6.4.1 Web Service Code Generators

The code generators are very easy to use. Usually these tasks are performed in a command line. In addition, within a couple of lines and a WSDL file path, it is possible to get code. The code generated parses the messages received and send it to the server. The following sections are brief introductions to the most popular software at this date available. First *gSOAP* is presented then *wsdl.exe* from the *.NET Framework*, then *Apache Axis* and finally a suggestion for another approach is made.

gSOAP

The *gSOAP* tools minimise the effort for building Web Services. *gSOAP* generates C or C++ code. To generate code a WSDL document must be provided. Using the *wsdl2h* tool a header can be generated. The following example shows how to use *wsdl2h* tool. The example is taken from *gSOAP* documentation.

```
wsdl2h -o calc.h http://www.cs.fsu.edu/~engelen/calc.wsdl
```

When the header document is generated, it is time to generate C++ code. For this purpose another tool *soapcpp2* is used. The following example shows how to use *soapcpp2*. This example is also taken from *gSOAP* documentation.

```
soapcpp2 -i -C -Iimport calc.h
```

The operation above generates client-side stubs and server-side skeletons.

For more information about *gSOAP* please visit the website [GSOAP_WSDL]

.NET

To generate code with .NET is very easy. For this purpose NET offers the tool *wsdl.exe* that can be accessed and executed from the SDK command prompt.

The syntax of the tool is:

```
wsdl [options] {URL | path}
```

To generate a Web Service client the WSDL document must be provided by specifying the path or URL of the WSDL document. Then the code is generated.

For more information about *wsdl.exe* please visit [NET_WSDL]

Apache Axis

The Apache Axis works in the same way as the others. It also provides tools to generate code, for example, *WSDL2Java* and *Java2WSDL*. The *WSDL2Java* tool generates Java code from the provided WSDL document. Apache Axis also provides a version to generate code in C++, but the process is the same.

For more information about Axis, please visit the following website [AXIS_WSDL]

Dynamic Web Service Client

Recently some investigation was developed concerning this topic. The idea of a dynamic Web Service client is to provide the WSDL file, and without any further compilation, the Web Service client changes itself. This can be done using the object-oriented concept. For the Integrated Concept, this solution is not very attractive because there is the need to re-compile the *VAN Communication DTM*

and the *VAN Device DTM*, because the parsers for the messages have to be updated. However, even these parsers can be dynamic, but for a prototype propose it is not interesting to put such an effort to provide a no-re-compilation solution.

6.4.2 Web Service Client Prototype Implementation

As described before the *WS_Client* component implementing the Web Service client can be generated with several different frameworks. From then, the *WS_Client* is integrated with the remaining *VAN Communication DTM* classes as described in 5.1.

The decision for choosing a specific framework to generate code depends on each person. The *WS_Client* is a very light component, so the decision to choose between the frameworks should go for how easy it is to integrate with the rest of the *VAN Communication DTM*. Therefore, it mainly depends on the platform and/or programming language used for the *VAN Communication DTM*. For the prototype, it was decided to use C# as the programming language for WP8, so the .NET framework is used to create the *WS_Client* for the prototype.

7 VAN Device DTM

7.1 Overview about components

The purpose of the VAN Device DTM is to run reused VAN components in a FDT environment with as few changes as possible. The VAN Device DTM behaves as FDT compliant DTM, providing (and requesting) all necessary FDT interfaces. Specially designed interfaces are used to integrate the reused VAN components, as introduced in deliverable D08.5-1 [D08.5-1]. Figure 26 gives an overview.

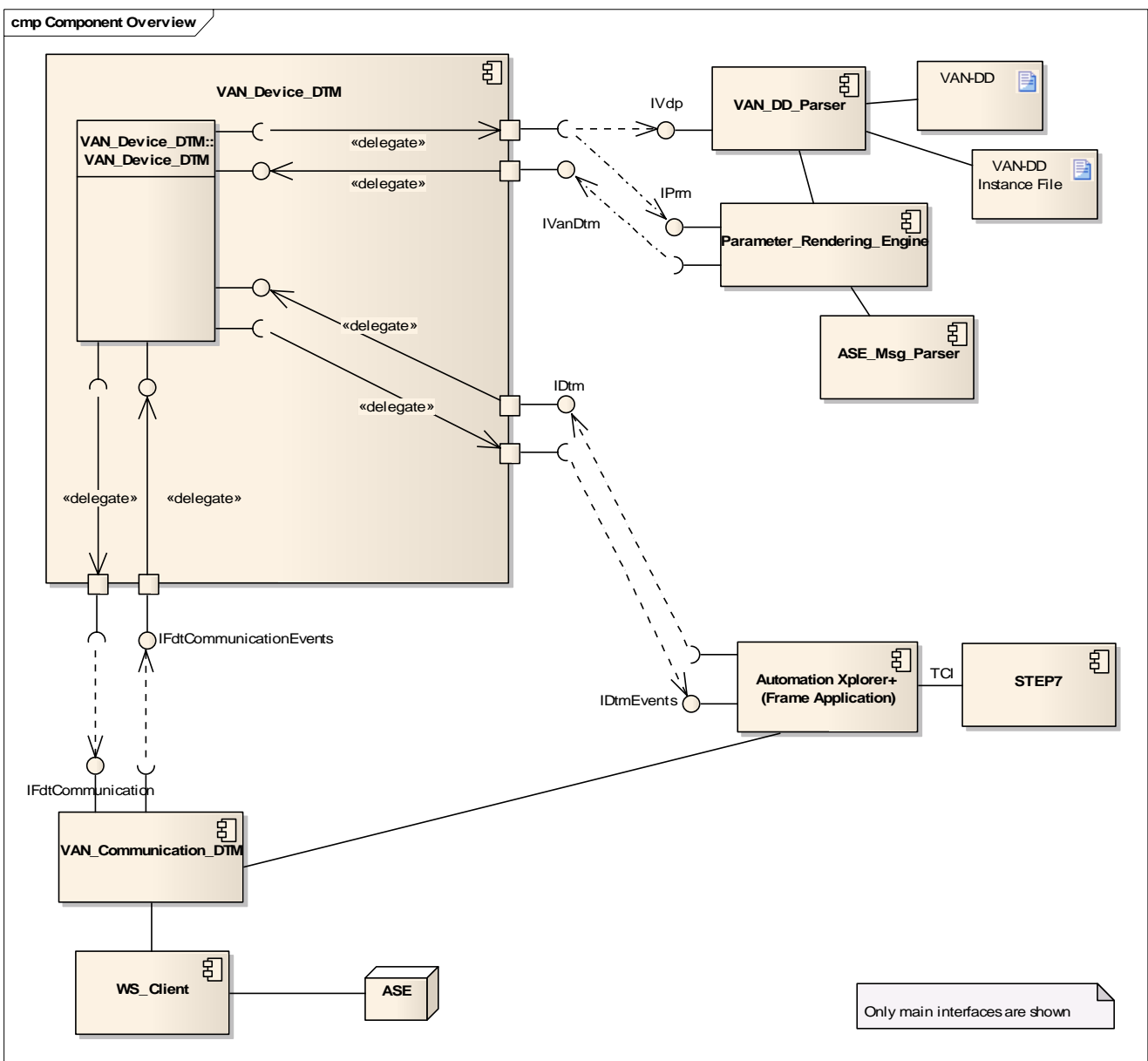


Figure 26: VAN Device DTM Architecture

The Device DTM works within the AutomationXplorer+, a FDT frame application (chapter 5). The main interface to be supported is `IDtm`, which is basically used for initializing, configuring, controlling, and releasing the Device DTM. Via `IDtmEvents` the Device DTM reports back to the Frame Application.

The VAN Communication DTM (chapter 6) provides the interface `IFdtCommunication` to the Device DTM, establishing the communication to the ASE via web services while using the VAN Communication schema. Via `IFdtCommunicationEvents` the VAN Communication DTM reports back to the Device DTM.

The Device DTM, the Communication DTM, and the Frame Application fulfil the FDT Specification 1.2.1 [FDT1.2.1]. The used interfaces of the VAN Device DTM are listed in chapter 6.2.

The reused components *Parameter_Rendering_Engine*, *ASE_Msg_Parser*, and *VAN_DD_Parser* are implemented and interact with each other as defined in [D08.5-1]. Their data (e. g. VAN-DD Instance files) are stored within the project file. This is not compliant to the FDT Specification which demands all (DTM) instance related data to be stored by the Frame Application alone and does not allow for private data storage mechanisms. For VAN prototyping purposes this incompatibility to the FDT Specification is tolerated. There are two specially designed interfaces necessary for interaction of the reused components with the Device DTM.

The interface `IPrm` is used by the Device DTM for managing the *Parameter_Rendering_Engine*. It is provided by the *Parameter_Rendering_Engine* and comprises methods as defined in Table 5.

Table 5: Methods of `IPrm`

Method	Description
New()	implicit, via standard methods of the control
Configure()	Determines which devices (name, vendor, version) has to be managed by the <i>Parameter_Rendering_Machine</i> . Additionally, a pointer to the Device Dtm interface <code>IVanDtm</code> is given.
PrepareToRelease()	The <i>Parameter_Rendering_Machine</i> saves all necessary data (may be asking the user for confirmation) in a persistent way. Success is reported back to the Device DTM using asynchrony using <code>OnPrepareToRelease()</code> .
Release()	implicit, via standard methods of the control
Show()	implicit, via standard methods of the control

The interface `IVanDtm` replaces the direct access to the *WS_Client*. Additionally it is used to report certain events back to the Device DTM. The interface `IVanDtm` is provided by the Device DTM and comprises methods as define in Table 6.

Table 6: Methods of `IVanDtm`

Method	Description
PutXmlString()	used after <code>decode()</code> method of the <i>ASE_Msg_Parser</i>
GetXmlString()	used before <code>encode()</code> method of the <i>ASE_Msg_Parser</i>
OnPrepareToRelease()	Reports success of <code>PrepareToRelease()</code> back to the Device DTM

The interface `IVdp` is used by the Device DTM for managing the `VAN_DD_Parser`. It is provided by the `VAN_DD_Parser` and comprises the following methods, defined in Table 7:

Table 7: Methods of `IVdp`

Method	Description
New()	implicit, via standard methods of the control
GetAllDevices()	Scans the predefined directory for VAN_DDs and delivers a list containing all available devices (name, vendor, version).
Release()	implicit, via standard methods of the control

A single directory is used to store all VAN-DDs, and the corresponding schemata. The name of the directory is always

`$AllUsersAppDataDir$\VAN\DDs\`.

The current value of the environment special directory can be asked at run time on each system.

A subdirectory structure may be defined later.

The executables `Parameter_Rendering_Engine`, `ASE_Msg_Parser`, and `VAN_DD_Parser` are implemented as .Net controls. Therefore these components can be reused either in form of a library or at source code level.

7.2 Implementation of FDT interfaces

The following subchapters define the level of implementation of the various DTM specific interfaces. It will be used the same semantic for the interpretation as defined in chapter 6.3.

7.2.1 Device DTM

Table 8: Level of Implementation for Device DTM Interfaces

Interfaces	Functions	Level of implementation
IDtm	Config	Full
	Environment	Full
	GetFunctions	Full
	InitNew	Full
	Invoke Function request	Dummy Implementation ignoring all in input parameters and return FALSE. No function is provided by the Device DTM.
	PrepareToDelete	Basic Channel assignment is not supported
	PrepareToRelease	Full
	PrepareToReleaseCommunication	Full

	PrivateDialogEnabled	Full
	ReleaseCommunication	Full
	setCommunication	Full
	SetLanguage	Basic Only English is supported
IDtm2	Environment2	Not-impl
	SetSystemGuiLabel	Not-impl
IDtmApplication	ExitApplication	Not-impl
	StartApplication	Not-impl
IDtmChannel	GetChannels	Dummy Returns only an empty list.
IDtmDocumentation	GetDocumentation	Dummy Return only a static standard string.
IDtmDiagnosis	Compare	Dummy Return always false.
	InitCompare	Dummy Return always false.
	ReleaseCompare	Dummy Return always false.
	Verify	Dummy Return always false.
IDtmImportExport	Export	Not-impl
	Import	Not-impl
IDtmInformation	GetInformation	Full
IDtmInformation2	GetDeviceIdentificationInformation	Not-impl
IDtmOnlineDiagnosis	Compare	Not-impl
	GetDeviceStatus	Not-impl
IDtmOnlineParameter	CancelAction	Not-impl
	DownloadRequest	Not-impl
	UploadRequest	Not-impl
IDtmParameter	GetParameters	Basic Return valid document with fixed set of parameters according to Table 1
	setParameters	Basic Accepts only the expected parameters.
IFdtCommunicationEvents	OnAbort	Full
	OnConnectResponse	Full

	OnDisconnectResponse	Full
	OnTransactionResponse	Full
IFdtCommunicationEvents2	OnConnectResponse2	Not-impl
IFdtEvents	OnChildParameterChanged	Dummy No parent-child relationship.
	OnLockDataSet	Dummy Method is empty
	OnParameterChanged	Dummy Method is empty
	OnUnlockDataSet	Dummy Method is empty
	CancelAction	Dummy Method is empty
IDtmSingleDeviceDataAccess	CancelRequest	Not-impl
	ItemListRequest	Not-impl
	ReadRequest	Not-impl
	WriteRequest	Not-impl
IDtmSingleInstanceDataAccess	GetItemList	Not-impl
	Read	Not-impl
	Write	Not-impl
IDtmActiveXInformation	GetActiveXProgId	Full
	GetActiveXGuid	Full
IPersistXXX	InitNew	Basic Method with no effect
	Load	Basic Method with no effect
	Save	Basic Method with no effect

7.2.2 Control for Dtm

Table 9: Level of Implementation for Control the DTM

Interfaces	Functions	Level of implementation
IDtmActiveXControl	Init	Full
	PrepareToRelease	Full

7.2.3 Communication DTM

For the interaction with the VAN Communication DTM, the level of implementation of interfaces as defined in chapter 6.1.3 are expected.

7.2.4 Frame Application

For the interaction with the FDT frame application, the level of implementation of interfaces as defined in chapter 5.4 are expected.

8 Conclusion

The scope of this deliverable was the specification of a prototype for an Engineering Tool for the Integrated Concept. The basic concepts and interfaces had already been identified in deliverable [D08.4-1] and in deliverable [D08.4-2]. These concepts were used in this deliverable to elaborate the specification of the Engineering Tool Prototype.

This is the first of two documents, which have to be worked out within task T8.6. Within the earlier tasks of work package 8, two feasible approaches for a VAN engineering tool have been identified: the Stand-Alone Concept and the Integrated Concept. The specification of a prototype for the Stand-Alone Concept is worked out in task T8.5, which was developed in parallel of this task.

The development of the specifications was based on a specific use case, here described as "Automation Example". The same example is used for work package WP5 of the VAN project and this use case can be engineered as described in this document.

The necessary enhancements for the used existing tools STEP7 and AutomationXplorer+ are described. Further, new components for the Integrated Concept were specified.

The specification of the Integrated Tool covers the role of a VAN-ECD. Thus, the engineering of VAN systems can be performed. This includes e.g. the import of VAN-DD Instance Files, the configuration and parameterisation of VAN devices and the sending of the engineering data to VAN devices via web services.

It should be mentioned, that a further advantage of this Integrated Tool is the usage of existing automation tools. So, the VAN parameterisation as well as the parameterisation and configuration of the devices, which has no VAN functionality, can be done inside the Integrated Tool.

The Integrated Tool prototype will be implemented in the next phase of task T8.6, after the completion of its specification. Further, to ensure the functionality of the tool, test cases must be specified, which represent the fundamentals of the verification and validation process.

A further task is to support the requirements that are requested from the other work packages especially with respect to the VAN Industrial Experimental Setup (IES), which will be represented at the end of the project. The IES represents an overall demonstrator with impacts of all work packages. Possible support of working package WP8 will be e.g. the supply of the engineering prototypes, the implementation of mechanisms to ensure the correct function of the IES and further the implementation of mandatory ASE schemas and VAN-DD Files.

After the implementation of the Integrated Prototype, which is done during the next phase of task T8.6, the next task within this work package will be started. Within the task T8.7, upcoming requirements from other work packages of the VAN project should be fulfilled, if it is possible and make sense to the existing implementation.

Glossary

Terminology

Automation System	The complete automation system, which is to be engineered, i.e. the union of all VAN domains.
Connection	In context of this deliverable a Connection is used to describe/create an end-to-end communication connection between distributed devices in different domains
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].
Engineering System	In the TCI terms the Engineering System is an engineering software used to engineer the complete system. The Engineering System calls the specific Device Tool to perform dedicated parameterization actions.
HW-Konfig	Hardware configuration tool for PROFIBUS and PROFINET
Interface	An interface defines the connection of an entity to its environment especially to other entities. It is the basis to create relations between the entities within the described system.
Module	Module is a part of a device
Network Device	A special device, which controls the communication on a network segment (network card, modem, etc) or which interconnects multiple network segments (hub, switch, router, gateway, wireless access point, etc).
Network Segment	Physical grouping → Physical part of a communication network with its devices. See table 1.3 of D02.2-1.
Project File	A file that is used by the Stand Alone tool to store the engineered information.
Transfermodule	Transfermodules define binary sequences of input or output ports of other devices. Each input or output port of any device is registered at a CPU under an address.
Transferunit	Is one or more bits as part of a transfermodule or the whole transfermodule

VAN-AP	A VAN Access Point defines an entity containing VAN-FS and connects VAN network segments but it does not contain an automation function or an automation application process.
VAN device	A VAN device consists of one or more application processes which define the device functionality and the connectivity to a VAN network. See chapter 2.2 of D02.2-1.
VAN Device Description	A VAN-DD is a formal description of configuration as well as parameterization capabilities in the VAN context. The specification provides the basic elements to describe a device for the VAN engineering.
VAN-DD Instance File	A VAN-DD Instance File is a XML based file and is used to store the configuration information for a concrete device within a VAN domain. It provides information about supported ASE objects, i.e. the network capabilities from the VAN point of view. It is an instance of VAN device description file and can be configured and parameterised respectively within a VAN-ECD.
VAN Domain Connection	This kind of connection is used to store general information about the relation between existing domains
VAN domain	Container for all device specific information about the domain, e.g. network topology, infrastructure communication connections; list of other VAN AP's or VAN AD's in the domain
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].

Abbreviations

ASE	Application Service Element
DD	Device Description
DTM	Device Type Manager
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
UML	Unified Modeling Language
UUID	Universal Unique Identifier
VAN	Virtual Automation Network
VAN-AD	VAN Automation Device
VAN-AP	VAN Access Point
VAN-DD	VAN Device Description
VAN-ECD	VAN Engineering Client for Automation Device
VAN-ECS	VAN Engineering Client for Automation System
WSDL	Web Service Description Language
WS	Web Services
XML	Extensible Markup Language

References

- [AXIS_WSDL] <http://ws.apache.org/axis/>
- [AX+] AutomationXplorer+ Engineering Application Software, <http://automationexplorer.phoenixcontact.com>, March 2008
- [D08.2-1]. Deliverable D08.2.1 of the VAN project *Engineering process concept and specification*. The VAN Consortium,2006
- [D05.4-1]. Deliverable D05.4-1 of the VAN project *Safety Mechanisms implementation in Process Industry and Manufacturing Industry; Prototype devices*. The VAN Consortium,2007
- [D08.3-1]. Deliverable D08.3.1 of the VAN project *Specification of product information model in general and of the mandatory product data*. The VAN Consortium,2007
- [D08.4-1] Deliverable D08.4-1 of the VAN project. *Specification of architecture and technologies for VAN engineering tool platform*. The VAN Consortium, 2007
- [D08.4-2] Deliverable D08.4-2 of the VAN project. *Specification of object model and tool interfaces for VAN engineering tool platform*. The VAN Consortium, 2007
- [D08.5-1] Deliverable D08.5-1 of the VAN project. *Specification of Engineering Tool Prototypes for Stand-Alone Concept*. The VAN Consortium, 2008
- [FDT1.2.1] FDT-JIG: FDT-Joint Interest Group, *Guideline – FDT Interface Specification*, V 1.2.1. FDT-JIG - Order No. 0001-0001-002, 2005
- [FDT_PB] FDT-JIG: FDT-Joint Interest Group *Guideline – Field Device Tool for PROFIBUS, Version 1.0, Annex to FDT Specification, Based on FDT Specification 1.2.1*, FDT-JIG – Order No. 0002-0003-00, 2005
- [FDT_PN] FDT Group: FDT Group AISBL, *Guideline – Field Device Tool for Profinet IO, V 1.0*, FDT Group – Document No. 0002-0008-000, 2007
- [GSOAP_WSDL] gSOAP Website <http://www.cs.fsu.edu/~engelen/soap.html>
- [PI] Profinet International PROFIBUS and PROFINET user organisation, <http://www.profibus.com/pi/>
- [STEP7] STEP7 Engineering Application Software, http://www.automation.siemens.com/simatic/industriesoftware/html_00/produkte/software-step7.htm, March 2008
- [TCI1] PROFIBUS PROFINET Specification: *Tool Calling Interface*, Version 1.0, October 2006, Order No: 2.602
- [TCI1.1] Specification for PROFIBUS and PROFINET: *Tool Calling Interface*, Version 1.1, July 2007, Order No: 2.602
- [TCI_A1] Annex 1 related to TCI V1.1 Specification: *Tool Calling Interface for PROFIBUS*, July 2007, Order No: 2.612
- [TCI_A2] Annex 2 related to TCI V1.1 Specification: *Tool Calling Interface for PROFINET*, July 2007
- [NET_WSDL] [http://msdn2.microsoft.com/en-us/library/7h3y5tb6\(vs.71\).aspx](http://msdn2.microsoft.com/en-us/library/7h3y5tb6(vs.71).aspx)