EUROPEAN COMMISSION

Thematic Priority:
SIXTH FRAMEWORK PROGRAM

Priority 2.5.3
INFORMATION SOCIETY TECHNOLOGIES
Unit G3 Embedded Systems

Project Acronym:

SOCRADES

Project Full Title:

Service-Oriented Cross-layer infRAstructure for Distributed smart Embedded devices

Proposal/Contract No: EU FP6 IST-5-034116 IP SOCRADES

Deliverable D7.3

Report of the state of the art in device support and maintenance

Status:	**Final**
Dissemination Level[1]:	**CONFIDENTIAL**
Date:	**27.09.2007**

Organization Name of the Lead Contractor for this Deliverable: **Loughborough University**

[1] See Annex A for explanation of Dissemination Levels, as defined in the DoW

Status Description:

Scheduled completion date[2]:	31.08.2007	Actual completion date[3]:	27.09.2007
Short document description:	This document describes the state of the art for commissioning and maintenance tasks of field devices.		
Author(s) deliverable:	Matthias Riedl (ifak) Christian Diedrich (ifak) Robert Harrison (Lboro) Radmehr P. Monfared (Lboro)	**Report/deliverable classification:** [x] Deliverable [ ] Three-Monthly Activity Report [ ] Six-Monthly Activity Report	
Partner ↓ Contributions ↓ Peer reviews	[ ] [ ] Schneider Electric [ ] [ ] ABB [ ] [ ] APS GmbH [ ] [ ] Boliden AB [ ] [ ] FlexLink Automation Oy. [x] [ ] Institut f. Automation und Kommunikation e.V. Magdeburg [ ] [ ] Kungliga Tekniska Högskolan	[x] [x] Loughborough University [ ] [ ] Luleå University of Technology [ ] [ ] Politecnico di Milano [ ] [ ] SAP AG [ ] [ ] Siemens AG [ ] [ ] Tampere University of Technology [ ] [ ] Jaguar Cars Ltd. [ ] [ ] ARM Ltd.	
Peer review approval :	[x] Approved [ ] Rejected (improve as specified hereunder)	Date:	26.09.2007
Suggested improvements:			

Version History:

Version:	Comments, Changes, Status:	Person(s):
0.1	Initial draft based on the format adopted for D1.2 by ifak	Matthias Riedl (ifak)
0.2	Adaptation of document structure according decisions of meeting	Matthias Riedl
0.3	Preparation first internal review	Matthias Riedl
0.4	Improvements and enhancements in configuration section	Rob Harrison, Radmehr P. Monfared, Matthias Riedl
0.5	Internal review comments	Rob Harrison, Radmehr P. Monfared
0.6	Improve structure	Matthias Riedl
0.7	Prepare final draft, comment from Lboro.	Matthias Riedl, Rob Harrison
0.8	Add this history, repair Fig. 20	Matthias Riedl
Final	Assessment of PCC	PCC members
Final	Assessment of the Project Coordinator	Armando Walter Colombo

[2] As defined in the DoW

[3] Scheduled date for approval

Executive Summary

Control system engineering and the instrumentation of industrial automation belongs to a very innovative industrial area as it is in the scope for cost reduction. That is why the costs for equipment and devices are already at a relatively low level. This trend is accompanied by a paradigm shift to digital processing in devices and digital communication between them.

The deliverable describes the historical development steps from analogue electronic devices connected via 4-20 mA or 24V technology to digital devices with industrial communication connections such as fieldbus and Web Service enabled devices, using Ethernet/TCP/IP. It derives from these steps the changes of the commissioning and system integration requirements and introduces the arising technologies to support device manufacturers and system integrators. All these integration technologies support the instrumentation tasks of field devices. Examples of these technologies are the Device Description Languages (provided e.g. by PROFIBUS, Device Net, Fieldbus Foundation), standardized interfaces such as OPC and FDT, control application integration using proxy function blocks written in PLC languages (e.g. in IEC 61131-3 languages) and vertical communication from field devices to SCADA, DCS and MES by means of XML. The reader will be familiar with up-to-date technologies used in field device engineering and instrumentation tools. The paper concludes with a formal model approach particular with a device model. This model provides an abstract view to the described technologies and makes the relations between them visible.

Field devices are integrated parts of the entire automation system. Therefore, new technologies arise to provide means for the device manufacturer, system integrators and system suppliers to design, commission, operate and maintain systems with a large variety of field device classes (Figure 1). These technologies are named integration/instrumentation technologies.

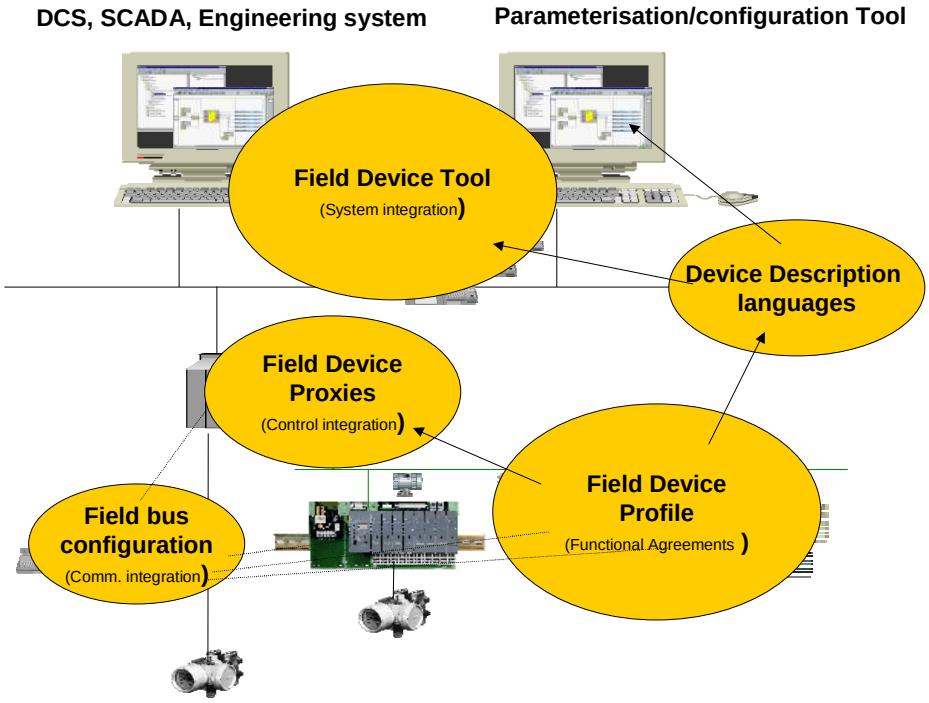


Figure 1: Integration technologies in automation systems

Field device profiles define common sets of functionality which are provided by devices of different manufacturers in an interoperable way. This is the semantic basis for the other technologies. The basis for all integration is configured communication systems which use according feature list languages. The Electronic Device Description Languages, the field device proxies and FDT/DTMs provide means to integrate device specific and innovative features in the system. The common trend of all described concepts is the use of web

based technologies mostly XML. This is an excellent starting point for multiple access to field device information from other applications than field device configuration and parameterization.

The deliverable D7.3 provides a focused technology overview to facilitate understanding of the device support related application engineering needs of distributed systems (feeding into Task 7.3). Furthermore the technologies described here will help to influence the approach used in Task 7.4 to simulate distributed automation systems. The biggest impact of D7.3 is, however, expected to be in helping to describe solutions for SOA-based commissioning and maintenance functions in Task 7.5.

Table of Contents:

List of Figures:

List of Tables:

1 State of the art of device support and maintenance for non-web service enabled devices

1.1 Configuration approach

Digital information computation in field devices and digital fieldbus communication lead to a change in the handling of automation systems in manufacturing and process control. The field devices now contain much more information than the 4-20mA signal. In addition, they carry out some functions which are/were originally programmed within the PLC or DCS. These field devices are also known as smart devices.

The consequence is a distributed system. The tools for the design and programming of control applications, commissioning as well as maintenance need access to both the data of the field device and/or an exact machine readable product description of the field devices including their data and functions. These device descriptions, the way from the design of the description to its use, are called device description technology. Using these device product descriptions it is necessary to integrate standard interface specifications and standardized industrial communication systems.

The following example of a transmitter shows the way from an analogue 4-20mA device to a smart fieldbus device. Other types of automation devices are going a similar way. Even digital, discrete and I/O field devices are much simpler, for example the 24V technology will also be replaced by fieldbus connected devices.

The transmitter in Figure 2 is composed of electronics which detect the specific measurement value (e.g. mV, mA) and which transform the detected signal into the standardized 4-20 mA. The adjustment to the specific sensor and wiring is done by trim resistors. Each transmitter is connected to the PLC by its own wires.

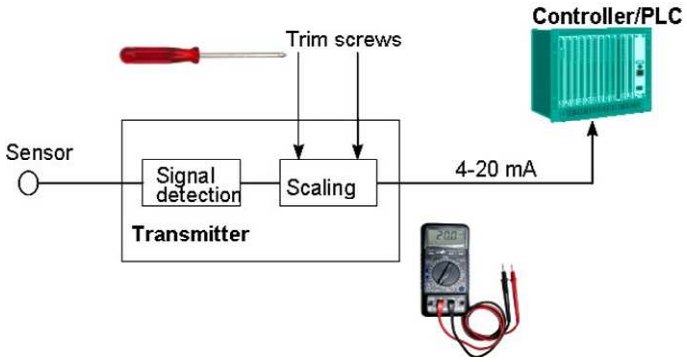


Figure 2: Structure of an analogue transmitter

Digital signal computation provides a higher accuracy. Therefore, the signal processing is carried out by micro processors (Figure 3). An analogue/digital unit and a digital/analogue unit transform the signals two times. The signal processing may be influenced by several parameters, which make the transmitter more flexible. These parameters have to be accessed by the operator. For this purpose, the manufacturer provides a local operator panel at the transmitter which consists of a display and very few buttons. PC tools provide more ergonomic solutions for the commissioning of such smart transmitters, which is necessary if they are more complex.

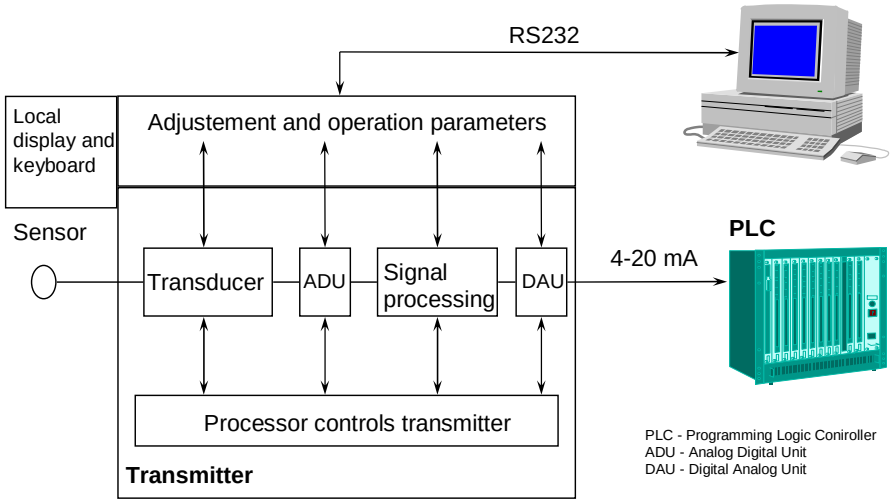


Figure 3: Structure of smart 4-20 mA transmitters

In principle, in smart fieldbus transmitters the digital/analogue unit at the end of the signal chain in the transmitter is replaced by the fieldbus controller. Now the measurement value remains digital. Two transformations - digital to analogue in the field device and analogue to digital in the PLC input card - as well as the analogue transmission is skipped. This increases the accuracy of the signal chain again. In addition, devices according to Figure 4 need specific communication commissioning using additional fieldbus configuration tools.

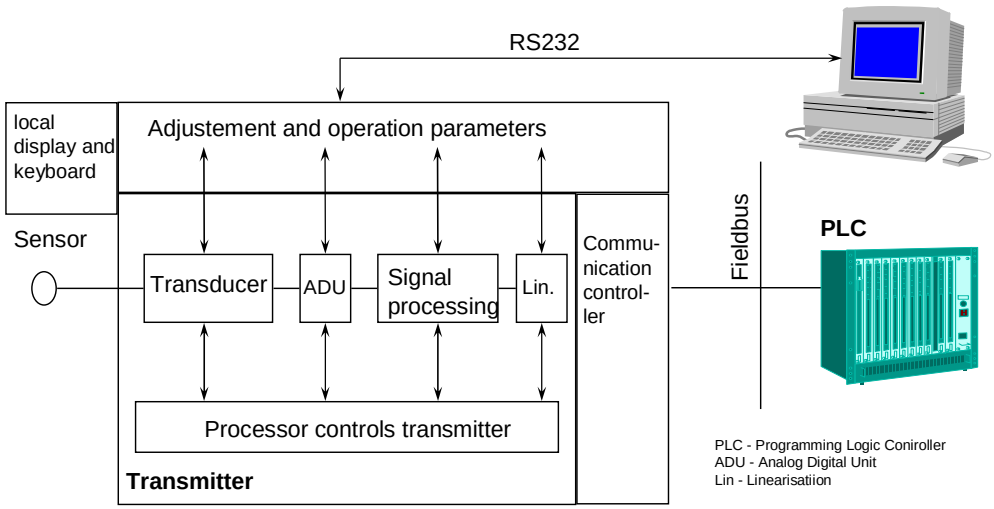


Figure 4: Structure of smart fieldbus transmitters

The smart field device parameterisation has moved from manual screwing over local device terminals to PC application software. Additionally, the communication system configuration has to be managed during the commissioning of the instrumentation.

The field device is typically integrated as a component in an industrial automation system. The automation system performs the automation related part of the complete application. The components of an industrial automation system may be arranged in multiple hierarchical levels connected by communication systems as illustrated in Figure 5.

The field devices are components in the process level connected via inputs and outputs to the process or the physical or logical sub-networks [19]. Programmable devices (e.g. programmable drives and remote I/O) and

router or gateways are also field devices. A communication system (e.g. a fieldbus) connects the field devices to the upper level controllers, which are typical programmable controllers or Decentral Control Systems (DCS) or even Manufacturing Execution Systems (MES). Since the engineering tools and the commissioning tools should have access to the field devices and to the controllers, these tools are also located in the controller level. The field devices may directly communicate via the fieldbus or the controller (Programmable Controller). In larger automation systems, an even higher level may exist which is connected via a communication system such as LAN or Ethernet. In these higher-level visualisation systems (HMI), DCS, central engineering tools and SCADA are located. Multiple clusters of field devices with or without a controller as described above may be connected with each other over the LAN or to the higher-level systems. Manufacturing Execution Systems (MES), Enterprise Resource Planning (ERP) and other Information Technology (IT) based systems can have indirect access to field devices via the LAN and the controllers or directly via routers.

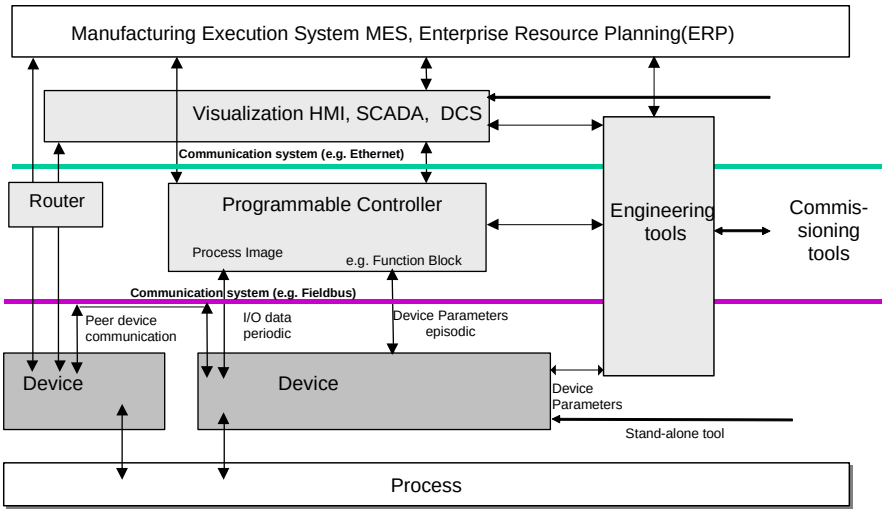


Figure 5: Typical automation configuration [19]

The result of this development is a large variety of PC based tools for planning, design, commissioning configuration and maintenance (Figure 5). This problem is also known in the home and consumer electronics. TV, CD player, video recorder and record player are gradually bought by a family, and these devices may be produced by different manufacturers. Each device is purchased with remote controllers. The user has to deal with different styles, programming approaches and outlines. This is unacceptable both, in consumer electronics and in the industrial environment. Of course, each manufacturer wants to be special regarding functions and features of its product and wants to provide special approaches to interact with the device. The end user gets products very suitable to its specific requirements, but he is faced with a broad variety of device interfaces.

The field devices are an integrated part of the entire life cycle of control systems starting with planning/design, including purchase, system integration/commissioning, operation and ending with maintenance. During each phase of the life cycle, specific information represented in different formats is used from the tools. Planning and design defines the requirements coming from the process which defines the type and the properties of the field device. The results are the pipe and instrumentation diagram (P&ID) in process control and electro-technical drawings (E-CAD) in factory automation combined with device lists. Purchase is done by requests of offers or directly by web market places. Commissioning carries out the parameterisation and configuration of field devices in connection with the programming of the programmable controllers. During operation there are mostly interactions between the controllers and the field devices. The field devices interact with special tools during maintenance. The view to field devices, i.e. the used subset of the entire information range, the format of the information (text in manuals, files, data

base entries, html pages) and the source of the information (paper, machine readable in the system, online from the device) differs between the lifecycle phases. Therefore, integration technologies deal with multiple technologies. MES are not in the scope of these collections of technologies. The main reason is that this paper concentrates on the functional aspects of field device integration. Even planning and design is only mentioned in certain sections because product data management does not yet have tight connections to the functional design of the control system.

1.2 Necessary information for device instrumentation

Today, the large number of different device types and suppliers within a control system project makes the filed device parameterisation and configuration task difficult and time-consuming. Different tools must be mastered and data must be exchanged between these tools. The data exchange is not standardised, therefore data conversions are often necessary, requiring detailed specialist knowledge. In the end, the consistency of data, documentation and configurations can only be guaranteed by an intensive system test.

The central workplace for service and diagnostic tasks in the control system does not fully cover the functional capabilities of the field devices, see Figure 6. Furthermore, the different device-specific tools cannot be integrated into the system's software tools. Typically, device-specific tools can only be connected directly to a fieldbus line or directly to the field device.

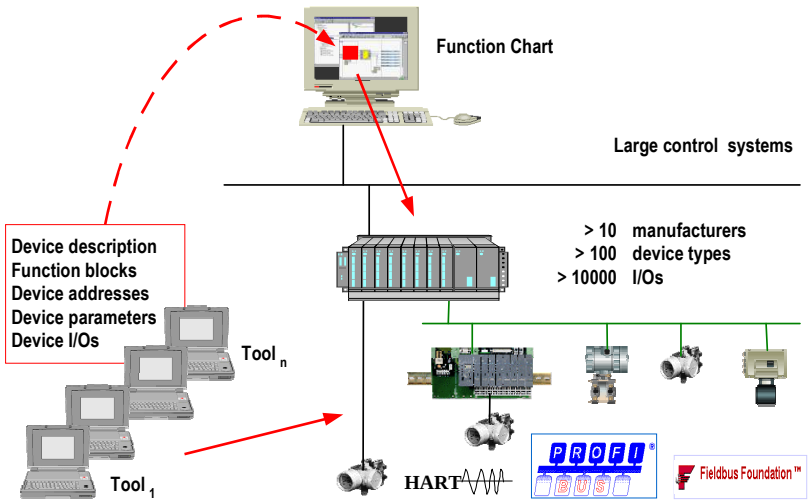


Figure 6: Different tools and multiple data input have determined field device integration to date

In order to maintain the continuity and operational reliability of process control technology, it is necessary to fully integrate field devices as a sub-component of process automation.

Field devices often have to be adjusted to their concrete application purpose, therefore, additional software components are necessary for parameterisation reasons. These components are necessary because local operator keyboards, with only a few bottoms and small displays, are not suitable to provide the complex parameterisation issues to the operator. In principle, the following cases of application can be found [6]:

- Devices with fix functionality and without parameterisation of its device application. These devices have to provide their communication properties which are only done according to descriptions such as PROFIBUS GSD and CAN EDS. A communication configuration tool checks all properties of all devices which are connected at one communication segment and generate the communication configuration in terms of the communication parameters such as baud rate and device addresses.
- Devices with only a small set of parameter data have to be fixed only once during the commissioning phase (e.g. minimum and maximum speed of a drive, calibration for a transmitter). This device should receive the data via fieldbus from the control station. Therefore, many fieldbus systems have added

parameterisation keywords in their communication related descriptions, such as PROFIBUS GSD and CAN EDS. Fieldbus configuration tools provide the possibility to edit the parameter values, the controller (e.g. PLC) ensures the persistence of the parameter data e.g. for a restart. This case of application does not need any additional parameterisation tools. The replacement of a device after failure can be easily done.

- Devices with a quite number of parameters and complex parameterisation means (e.g. transmitters, actuators in the process control field) have, for a long time, had local terminals and/or their own commissioning tools. Tools which are able to parameterise many different types of devices from different manufacturers have been established on the market. Well-known languages are HART DDL (Device Description Language [13]), Fieldbus Foundation DDL [12] and the PROFIBUS EDD (Electronic Device Description [24]), which belong to the same language family [16]. These languages are characterised by off-line parameterisation, device specific guidance by the operators and extensive consistency checks. The management of the persistence data is directly related to parameterisation tools, therefore a replacement of these devices without these tools is not possible.
- Devices with complex parameterisation sequences and complex data types (e.g. graphical information representation or video representation) can not be described by the languages mentioned. Therefore, these devices could need device (e.g. Laser scanner) specific commissioning tools.

In addition to the parameterisation of field devices, the functions of the smart field devices have to be integrated in the control application. This is not a topic of the vertical integration of field devices and not the scope of this text. However there are multiple concepts for it such as the IEC61499 [15], PROFInet [25] and IDA [14] standards as well as the proxy concept. The proxy concept is state-of-the-art; the other standards are not yet available on the market.

1.2.1 Configuration of communication

1.2.1.1 GSD language

A communication segment is an application specific collection of devices which are connected to it also known as communication configuration. The baud rate, device addresses and special communication timing parameters have to be adjusted according to the collection of devices and their properties. Therefore, field devices provide communication feature lists in a machine readable format. Each device is purchased with such a list which is used by a communication configuration tool to generate the possible or optimal communication configuration. Nearly each fieldbus organisation has specified its own feature list "language". Examples are the GSD of PROFIBUS, EDS of CANopen and DeviceNet as well as the FDCML of Interbus S.

To describe the principles we use the PROFIBUS GSD example. The abbreviation GSD stands for German Device Data Base (GeräteStammDaten). There is no translation of the abbreviation at all. The GSD describes the communication features and the cyclic data of simple devices such as binary and analogue I/O. Communication features are potential baud rates, communication protocol time parameters and the support of PROFIBUS services and functions (e.g. remote address assignment).

Simple I/O devices (also known as Remote I/O or intelligent clip) are mostly modular. A basic device with power consumption and a central processing unit has several slots, offered to plug in modules with different signal qualities (analogue, binary) and quantities (e.g. 2, 4, 8, 16 channels). According to the chosen modules, a different set of data has to be transferred in the cyclic transfer between the PROFIBUS master and slave devices. In other words, every module contributes with a specific set of data to the cyclic telegram. If we consider that the configuration tools of the PROFIBUS master devices (PLC mostly) have to be used device manufacturer independent, an unambiguously and manufacturer independent description of the modules is necessary. This is the second main task of the GSD description.

The GSD is a list of keywords accompanied by their value assignments for each feature of the PROFIBUS DP protocol which is configurable. This list is accomplished by a device and GSD file identification. Figure 6 shows how a GSD file looks.

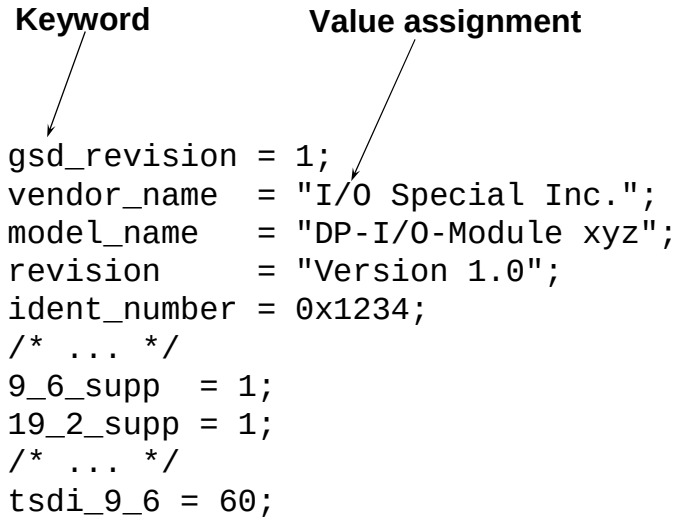


Figure 7: Example GSD file subset

Modules are represented in the GSD by the keyword „Module“. Between the keyword `Module` and the `End_Module` keyword, the declaration of the contribution of the module to the cyclic data telegram is specified in terms of binary codes, the so called Identifier Bytes. Each module has a name which is printed at the configuration tool screen. If the module is chosen, the specified Identifier Bytes will be concatenated to the configuration string, which is transferred from the master to the slave during the start up of the cyclic data transfer (CFG service of PROFIBUS-DP [23]). In addition, the range and the default value of variables can be configured which is supported by the variable declaration possibility of GSD. This data specified in the module GSD construct is transferred with the so called PRM service (PRM from parameterisation) of PROFIBUS DP ([23]).

1.2.1.2 GSD tool set

All PROFIBUS DP manufacturers are obliged to sell these devices with a GSD file which is checked by the PROFIBUS User Organisation (PNO). The GSD file is using as transfer syntax of the feature list ASCII text as shown in Figure 7. The PNO offers a special GSD editor which guides the engineer during the GSD development (Figure 8). The engineer chooses the features of the device with mouse click from the superset of all existing PROFIBUS-DP features. The editor generates the syntactic right GSD ASCII file. This file is not compiled.

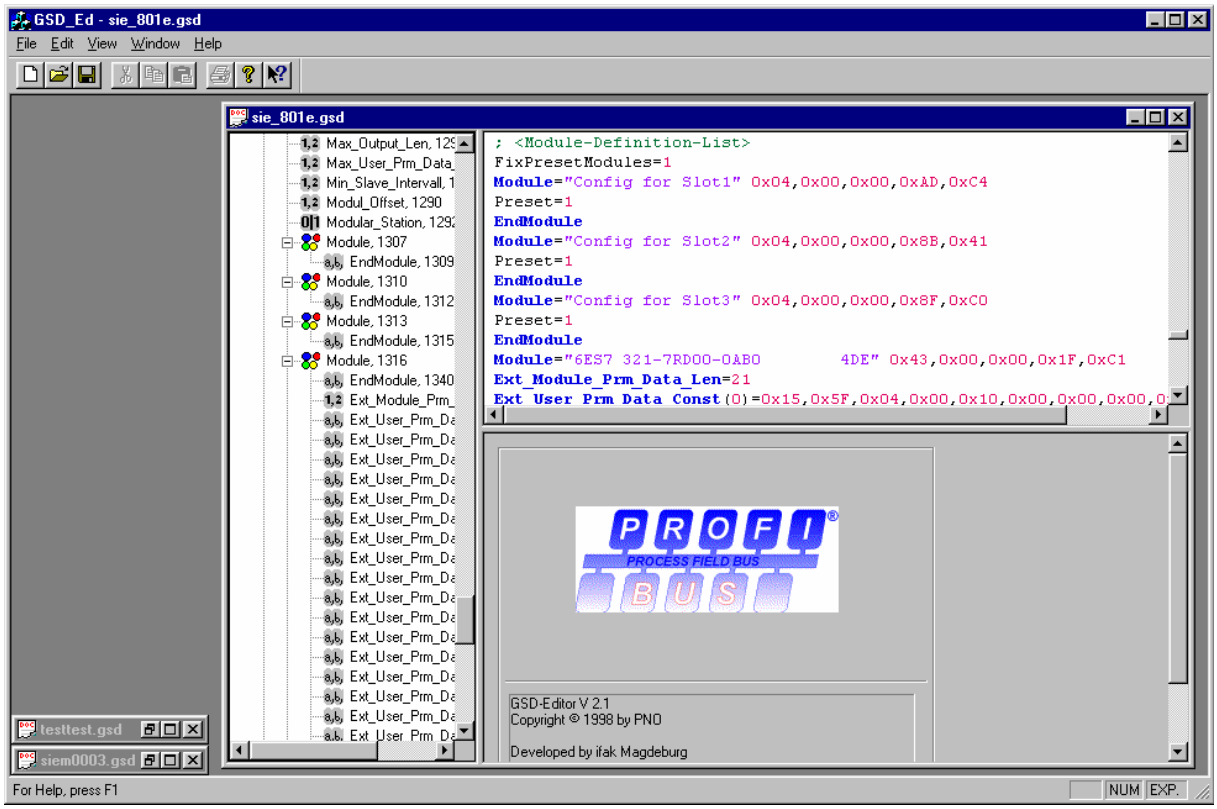


Figure 8: GSD editor snapshot

The PROFIBUS User Organisation has started an activity to provide an XML based transfer syntax of the GSD information which can be additionally used by configuration tools in future.

The configuration tools read these GSD files and present the specified device features for communication configuration and adjustment of some application parameters to an operator. The operator has to accept or modify the configuration parameter. The result of this configuration process is the configuration and parameterisation parameters which are loaded in the PLC, i.e. the PROFIBUS DP master class 1 (Figure 9).

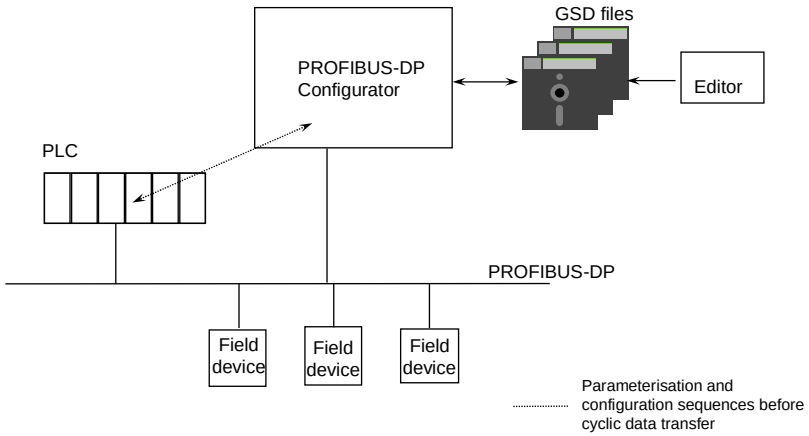


Figure 9: Tools and handling of GSD

The handling of GSD is very simple and needs no specific training. It matches exactly the needs of simple I/O with no or less application parameterisation.

1.2.2 Configuration of application

1.2.2.1 Configuration of legacy devices

One accepted approach in order to configure application specific parameters is the usage of device descriptions.

Two kinds of users can be distinguished for the task of application configuration:

- the end-user or operator of a plant or machine
- the system integrator

For the end-user of a distributed control system, the most important thing concerning device description is its transparency. The end-user wishes to see only the graphical user interface of a device represented in the SCADA-software or other Human Machine Interfaces (HMI). Therefore, the electronic device description has to be constructed following the plug & play concept. Furthermore, exchanging a device in an application is not required to lead to a big engineering task, however it is required to be done simply and securely by a technician. The device description has to support this.

In contrast to the end-user, the system integrator, who does the engineering, instrumentation and documentation have different targets to meet. His goal is to reduce the engineering time for solving interoperability problems but to enhance the engineering effort to design the optimal application with regard to the quality of the product to be produced with the process control system. The effectiveness of his work, comprising all pre-producing phases of the plant/machine life cycle, depends directly on the support of a well-defined, standardized and complete machine-readable description of the devices he has to deal with.

The conclusions from this analysis are, that the Device Description Language has to be designed mainly from the system integrator's point of view, because he has to deal directly with the electronic device description. The fieldbus community developed some approaches to exchange data by electronic means. This includes Device Descriptive Languages (DDL; e.g. for HART, FF, and Electronic Device Description for PROFIBUS). Main features of this description technology, including the according language, can be summarized as follows:

- A device description (in terms of a file) is delivered by the device vendor together with the device
- A device description is used in the engineering process of the distributed control system, supporting planning, commissioning, operation, diagnostics and maintenance
- There are different device description presentations possible; source in a human readable way and binary format
- A device description is mostly stored on disc and can additionally be stored within the device (transport via fieldbus)
- A device description is nearly independent from the underlying fieldbus system
- A device description is used to describe information identifying each item and defining relationships between them (hierarchical, relational)
- A device description covers language elements for presentation within a HMI and for communication access
- A device description file represents a static description, i.e. the declarative part of the device, therefore only the external interface/behaviour of the device is described (no interest in internal code)

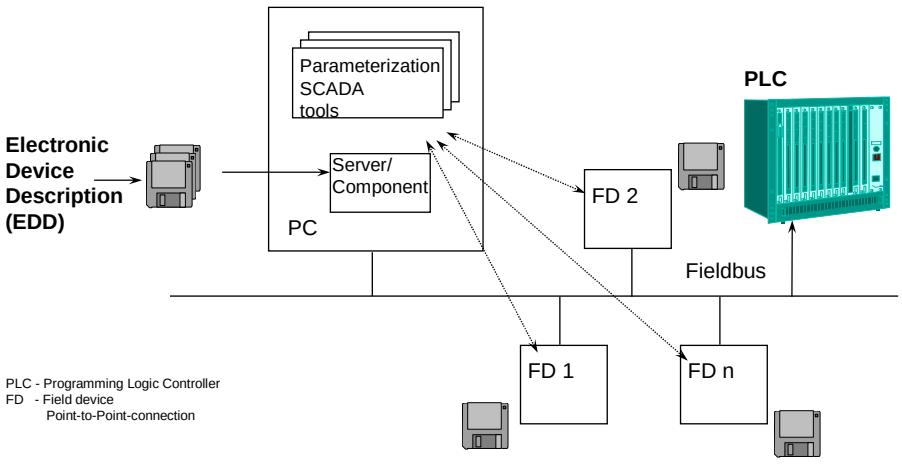


Figure 10: Using Electronic Device Description in configuration tools

A parameterization or configuration tool needs special adaptation to functions and parameters of each device which should be commissioned or visualised (Figure 10). The Electronic Device Description contains all device functions and parameters. The functions and parameters differ between the devices, but they are described with one and the same defined language. The tools accept this language and adapt themselves to the described functionality. Adaptation means, modification of the screen outline (e.g. content of the menus and bars) and modify the interactions with the devices. Generally speaking, the devices will be purchased with a disk containing their own device description, and the tools get their specific configuration by reading and interpreting this device description. The main parts of the Electronic Device Description are:

- Description of variables and parameters including their attributes (Process Variable - PV with label for screen prints, e.g. Level hydrostatic, data type floating point, access right read/only, default value = 0)
- Presentation of variables in HMI tools (e.g. PV with label „Temperature“)
- Guidance of the operator during commissioning (e.g. order of menu entries at hand held terminals or PC tools)
- Table of content of all visible elements of the device
- Device configuration parameter (e.g. identifier for plug-in modules of modular devices)
- Communication configuration parameters (e.g. baud rate = 500 kBaud)
- The following example gives a short look-and-feel of the language:

EDD [16] is a formal language, used to describe completely and unambiguously, what a field instrument looks like when it is seen through the "window" of its digital communication link. EDD includes descriptions of accessible variables, the instrument's communication related command set, and operating procedures such as calibration. It also includes a description of a menu structure which a host device can use for a human operator. The EDD, written in a readable text format, consists of a list of items ("objects“) with a description of the features ("attributes“ or "properties“) of each. Some example fragments from an (imaginary) flow meter EDD are shown in Figure 11.

```
VARIABLE v_low_flow_cutoff
{
  LABEL “Low Flow Cutoff”;
   HELP “The value below which the process variable will indicate
         zero,to prevent noise or a small zero error being
```

```
        interpreted as a real flow rate“;
  TYPE FLOAT
  {
    IF v_precision = high
      DISPLAY_FORMAT „4.6f“;
    ELSE
      DISPLAY_FORMAT „4.2f“;
  }
  CONSTANT_UNIT „%“;
  HANDLING READ & WRITE;
  }

MENU m_configuration
{
  LABEL „configuration“
  ITEMS
  {
    v_flow_units, /* variable */
    range, /* dialog */
    v_low_flow_cutoff,  /* variable */
    m_flow_tube_config, /* menu */
    m_pulse_output_config     /* menu */
  }
}

COMMAND write_low_flow_cutoff
{
  INDEX 37;
  SLOT 1;
  OPERATION WRITE;
  TRANSACTION
  {
    REQUEST
    {
      v_low_flow_cutoff
    }
    REPLY
    {
    }
  }
  RESPONSE_CODES
  {
    0.  SUCCESS,  [no_command_specific_errors];
    3.  DATA_ENTRY_ERROR,     [passed_parameter_too_large];
    4.  DATA_ENTRY_ERROR,     [passed_parameter_too_small];
    6.  MSC_ERROR,      [too_few_data_bytes_received];
    7.  MODE_ERROR,     [in_write_protect_mode];
  }
}
```

Figure 11: Device Description example

The example contains the parameter (or variable) `v_low_cutoff` (`VARIABLE` keyword) with its features

- Label and help (text to be printed at the PC screen of operator panel for the operator)
- Data type (in this case float with a print format of 6 or 2 digit after the dot)
- Fixed unit of %
- Access rights read and write

To write the variable, the command (`COMMAND` keyword) contains the necessary information for the parameterisation of the communication services such as:

- Index, Slot (access address of the variable within the device)
- Service (in this case write) using the keyword OPERATION
- The reference to the variables which have to be transmitted (in this case `v_low_flow_cutoff` to the field device (keyword `REQUEST`) and the response code (mandatory for all commands and generated by the communication system, key word REPLY) using the keyword `TRANSACTION`
- Operator message of the response code (defines the text to be printed out for the operator) by means of the keyword `RESPONSE_CODE`

The menu `m_configuration` defines that the variables and sub-menus which are declared within this menu are at the same visualisation level at the screen.

The major benefit of EDD for suppliers is that it decouples the development of host and field devices. Each designer can complete product development with the assurance that the new product will interoperate correctly with current and older devices, as well as with future devices not yet invented. In addition, a simulation program can be used to test the user interface of the EDD, allowing iterative evaluation and improvement, even before the device is built.

For the user, the major benefit is the ability to mix products from different suppliers, with the confidence that each can be used to its full capacity. Easy field upgrades allow host devices to accept new field devices. Innovation in new field devices is encouraged. The EDD is restricted for the description of a single device and use in a mostly stand alone tool.

Software tools for automation are very complex and implements a lot of know-how. The number of sold products is relatively low in comparison with office packets. The definition of standardised Device Description Languages increases the potential users of the tools and speed up the use of fieldbus based automation.

1.2.2.2 Field device system integration

Common for the technologies described in 1.2.1 and 1.2.2 is the limited scope of data in so called 'stand-alone' tools. The engineering and supervisory system (Figure 5) needs data from all devices and components in order to provide a non-interrupted information flow between the tools.

In order to maintain the continuity and operational reliability of process control technology, it is therefore necessary to fully integrate field devices as a sub-component of process automation [10]. To resolve the situation, the German Electrical and Electronic Manufacturers' Association (ZVEI) initiated a working group in 1998 to define a vendor-independent Field Device Tool (FDT) architecture. Recently the specification will be maintained and refined inside the FTD Group [11]. This FDT concept defines interfaces between device-specific software components (DTM - Device Type Manager) supplied by device manufacturers, and engineering systems supplied by control system manufacturers. The device manufacturers are responsible for the functionality and quality of the DTMs, which are integrated into engineering systems via the FDT interface. With DTMs integrated into engineering systems in a unified way, the connection between engineering systems (e.g., PLC applications) and inconsistent field devices becomes available. The FDT specification specifies what the interfaces are, not the implementation of these interfaces [10]. Figure 12 shows the FDT architecture. From the Figure, we can see that DTMs act as bridges between the frame-application and field devices. Several articles about FDT summarize the features of FDT, e.g. [28].

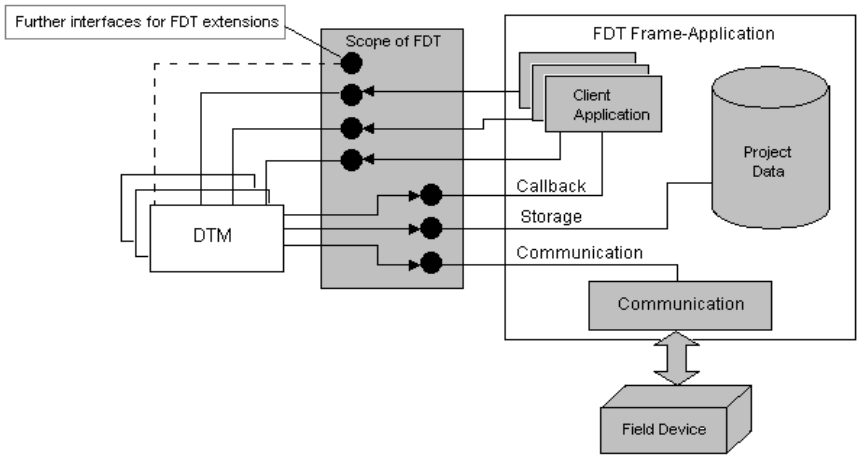


Figure 12: The architecture of Field Device Tools

FDT Frame-Application

As one component of the FDT structure, the frame-application is supposed to manage data and communication with the device and embedded DTMs. According to the environment, a DTM running in the frame-application can be an engineering tool, or even a web page. A frame-application is mainly considered as an engineering environment which integrates field devices and their configuration tools and controls DTMs within a project. The project is a logical object to describe the management and controls, at least in the lifetime of device-instances in terms of the according DTM within a frame-application [5]. In the view of a DTM, interfaces such as `IPersistPropertyBag` (Connection Storage in Figure 12), `IFdtCommunication` (Connection Communication in Figure 12) specified in [5], determine what the frame-application can do.

FDT Device Type Managers

The DTM is the key concept of vertical field device integration, with each DTM representing one field device. A DTM is an executable software component for a Windows PC which is purchased together with the device. Additionally it is possible that there are DTMs which represent classes of devices (based on field device profiles) or generic DTMs which are parameterized by device descriptions such as EDD. The DTM is characterised as follows:

- it supports COM/DCOM interfaces detailed by the FDT specification
- it executes all business rules of the device
- it contains all user dialogues (including multi language support system)
- it performs complex algorithms e.g. calibration
- it reads/writes variables from/to the device
- it support device specific diagnosis functions
- it knows all device specific data for the communication with the device
- it stores the instance data of the device
- it provides the device specific documentation
- it has no direct connections to other devices
- it supports one or multiple device types
- it does not communicate directly with the device
- it stores the data outside the DTM in a system component linked by the FDT framework

DTMs are installed in engineering tools or other applications (both are frame applications), which manage the field device instances. The frame applications install the DTMs and provide means for the communication and data storage. During installation the DTM gets all the necessary information about its environment (i.e. links to communication, data base components). This is the pre-requisite to control the access conditions and rules of the DTM. In detail, the frame application provides the following information:

- actual user (actor in the sense of UML)
- actual operation phase

The following users are defined:

- Observer

  The observer is a person who is only able to monitor the devices processed.

- Operator

  The operator is able to influence the running process additional to the possibilities of the observer.

- Maintenance operator

  The maintenance operator is able to perform maintenance work, e.g. replace devices or device calibration.

- OEM Service operator

  The OEM Service operator is able to perform manufacturer specific maintenance work, e.g. download new firmware revision. The OEM Service operator is also known as the “abstract user“.

- Planning Engineer

  The planning engineer may use additional functions to the normal maintenance one (i.e. maintenance operator functions).

- Administrator

  The administrator may add or delete software components within a FDT system. He is also known as “abstract user“.

The following operation phases are defined:

- Engineering

  Planning and configuration of a site. Online access to the devices and components are not possible.

- Commissioning/shop floor

  Installation of the site and devices. Download of project data in the devices and components. Programming and diagnosis of the devices, adjustment of the parameters.

- Runtime

  The site is in full operation. Access to the configuration and parameterization data is limited. Defect devices can be replaced. Interaction with the field devices is mostly to exchange process values and diagnosis data.

There are multiple ways to implement DTMs:

- Based on device descriptions (e.g. EDD, GSD) within a generic DTM
- Extension of a stand alone tool to a DTM
- Specific manual implementation (e.g. using C++, VB)

- Based on device descriptions (e.g. EDD, GSD) through translation of the descriptions to common languages (e.g. C++, VB)

The decision for a specific implementation technology depends on several aspects, for example device complexity, existing components or descriptions and existing know how.

FDT Interfaces

From the above parts, interfaces of frame-application and DTMs are mentioned and shown as the bridges between frame-application and DTMs. From the outside of an object, we can see only these interfaces that are specified according to their special functionalities but their implementation remains invisible and encapsulated. As an example, the communication between device and DTM shows the basic principles.

The FDT specification supports the communication protocols of HART and PROFIBUS (PROFIBUS-DPV0 / PROFIBUS-DPV1). Communication and gateway DTMs have so called channels, which provide access to the communication using software protocols. Figure 13 shows a device DTM, a communication DTM and the channel between both components. `IFdtCommunication` is the name of the interface and `TransactionRequest()/OnTransactionResponse()` are software protocol services.

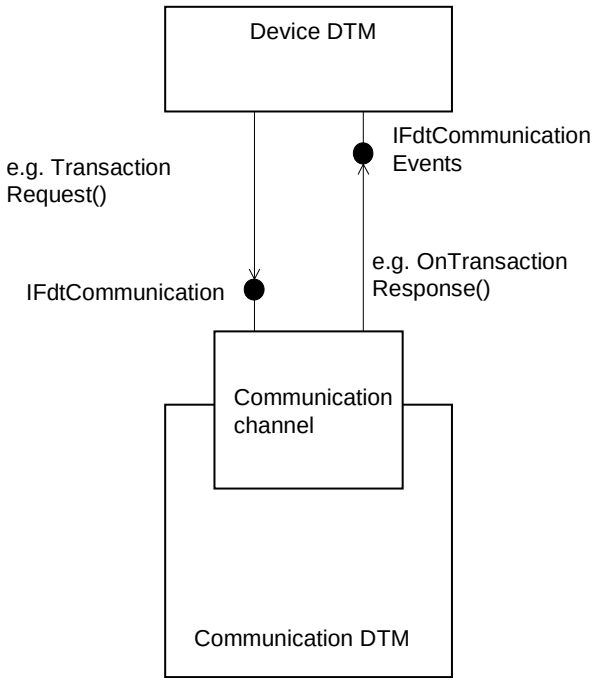


Figure 13: Connection of the communication channel and device DTM

The communication channel is part of the communication DTMs. Its instantiation and assignment to the device DTM is performed by the frame application using the following sequence (Figure 14):

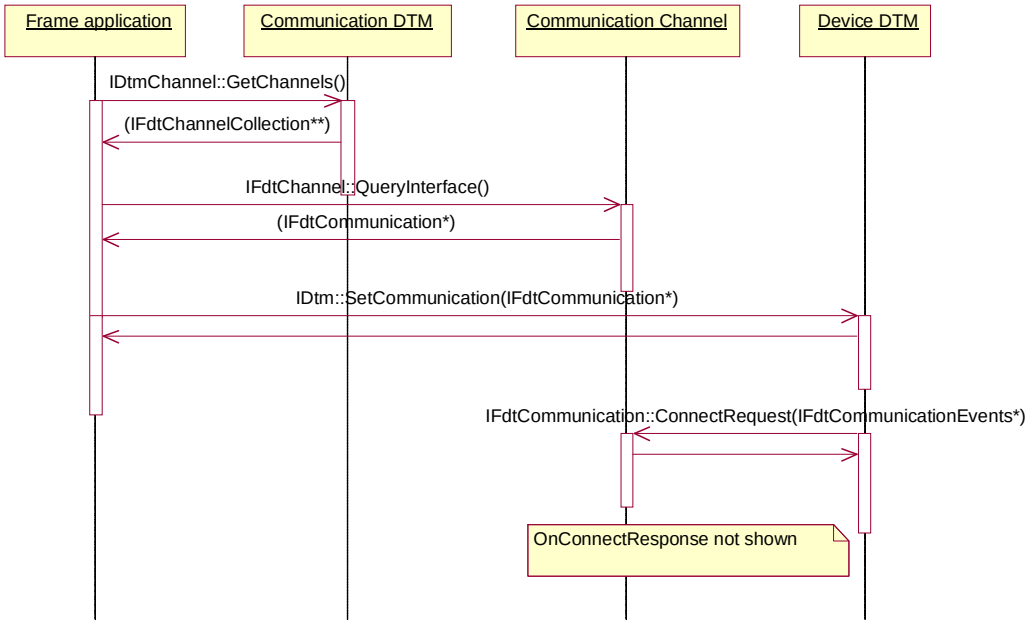


Figure 14: Sequence diagram: Initiate communication hierarchy

The interface `IDtmChannel` of the communication DTMs provides details of the communication channels. The frame application chooses a communication channel and its Interface `IFdtCommunication` using a Query interface service. The device DTM gets the result (pointer to `IFdtCommunication` interface) using the method IDtm::SetCommunication(). Then the device DTM can use the communication using the `IFdtCommunication` interface.

The communication DTMs are responsible for the sequence of services between the device and itself. The device DTM has no detailed knowledge about it because this is dependent on the communication configuration. This includes state machines, queues for service requests, coding and decoding of software service primitives as well as internal resource management. The responses of service requests are handled by the `IFdtCommunicationEvents` interface of the device DTM.

The main purpose of the connection between device and the communication channel is the productive data transfer. This is shown in Figure 15:

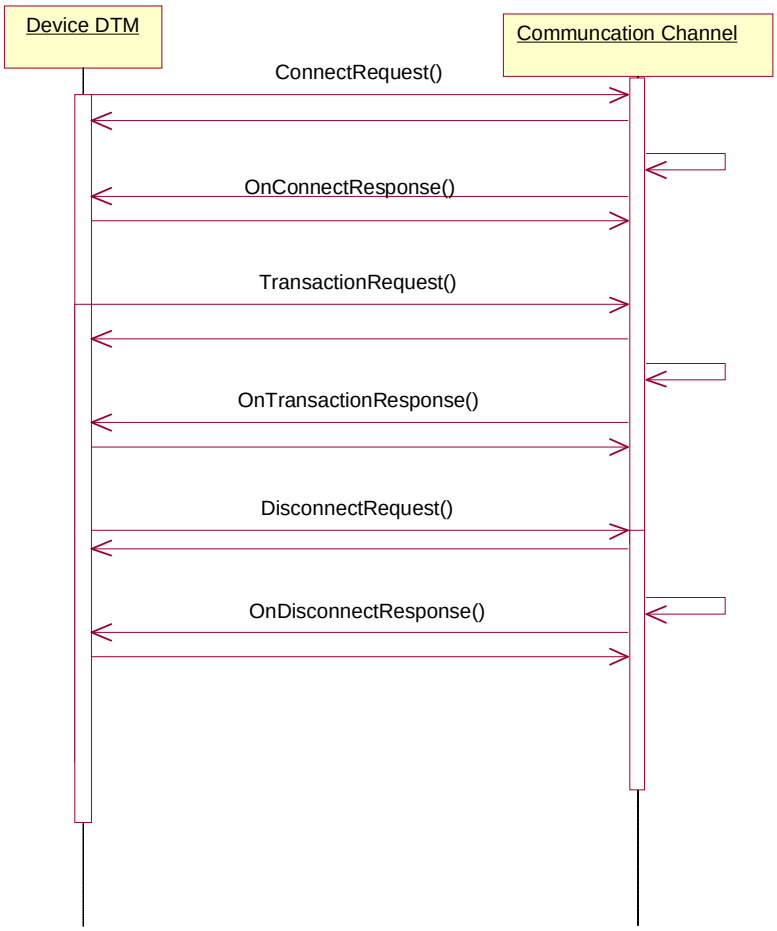
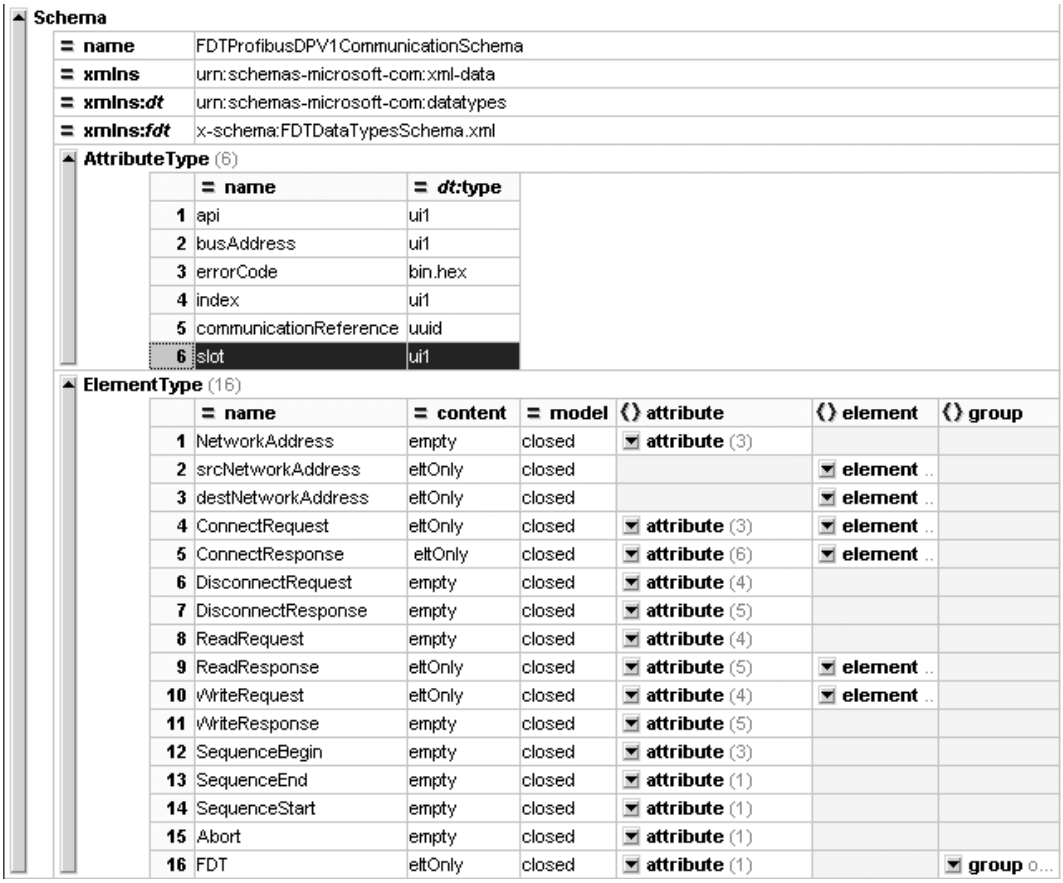


Figure 15: Sequence diagram: Productive data exchange

The productive data transfer is started with the connection initialisation. This is followed by the cyclic communication. The connection abort closes the connection. The communication channel takes all own responsibility to perform the service interactions. FDT does not describe how to handle the communication controller interface nor the fieldbus protocol. This is hidden in the general service.

The interfaces between the FDT components, in our example between device and communication DTM is implemented as exchange of XML documents. The XML schema used for PROFIBUS-PROFIBUS-DPV1 is partly shown in Figure 16:



Schema

= name	FDTProfibusDPV1CommunicationSchema
= xmlns	urn:schemas-microsoft-com:xml-data
= xmlns:*dt*	urn:schemas-microsoft-com:datatypes
= xmlns:*fdt*	x-schema:FDTDataTypesSchema.xml

AttributeType (6)

	= name	= *dt:*type
1	api	ui1
2	busAddress	ui1
3	errorCode	bin.hex
4	index	ui1
5	communicationReference	uuid
6	slot	ui1

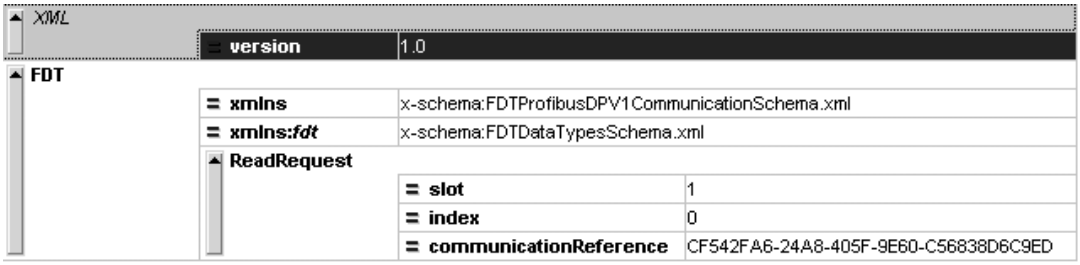
ElementType (16)

	= name	= content	= model	{} attribute	{} element	{} group
1	NetworkAddress	empty	closed	attribute (3)		
2	srcNetworkAddress	eltOnly	closed		element ..	
3	destNetworkAddress	eltOnly	closed		element ..	
4	ConnectRequest	eltOnly	closed	attribute (3)	element ..	
5	ConnectResponse	eltOnly	closed	attribute (6)	element ..	
6	DisconnectRequest	empty	closed	attribute (4)		
7	DisconnectResponse	empty	closed	attribute (5)		
8	ReadRequest	empty	closed	attribute (4)		
9	ReadResponse	eltOnly	closed	attribute (5)	element ..	
10	WriteRequest	eltOnly	closed	attribute (4)	element ..	
11	WriteResponse	empty	closed	attribute (5)		
12	SequenceBegin	empty	closed	attribute (3)		
13	SequenceEnd	empty	closed	attribute (1)		
14	SequenceStart	empty	closed	attribute (1)		
15	Abort	empty	closed	attribute (1)		
16	FDT	eltOnly	closed	attribute (1)		group o...

Figure 16: Schema PROFIBUS-PROFIBUS-DPV1

This schema describes the "software“ service primitives of the PROFIBUS-PROFIBUS-DPV1-protocolls. The schema starts with its Name, followed by external data type definitions (2* Standard Microsoft, 1* FDT basic data types). Then some attribute types follow. Figure 16 shows only a subset (6). The Element-Type section contains the "software“ service primitive (e.g. `ConnectRequest`, `ReadResponse`). These service primitives are used by the methods of the interfaces `IFdtCommunication` and `IFdtCommunicationEvents`.

An XML document based on this schema is shown in Figure 17 (Read-Request):



XML

= version	1.0

FDT

= xmlns	x-schema:FDTProfibusDPV1CommunicationSchema.xml
= xmlns:*fdt*	x-schema:FDTDataTypesSchema.xml

ReadRequest

= slot	1
= index	0
= communicationReference	CF542FA6-24A8-405F-9E60-C56838D6C9ED

Figure 17: Example for XML document PROFIBUS-PROFIBUS-DPV1Read-Request

This XML document firstly references the used XML schema. It is followed by the `ReadRequest` “software“ service primitive. It contains the attribute Slot (address of object which is intended to be read from the device), Index (sub-address of the object) and communication reference (address of the device). The `ReadResponse` “software“ service primitive is based on the same schema. Its XML document is shown in Figure 18.

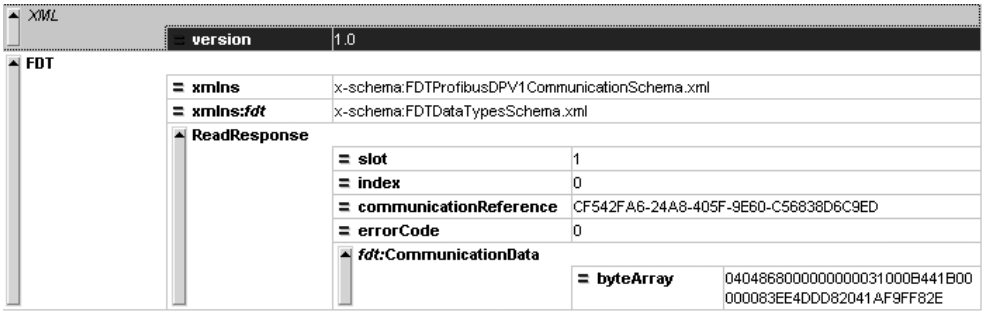


Figure 18: Example for XML document PROFIBUS-PROFIBUS-DPV1Read-Response

Both the communication and device DTM have to code its sending "software" service primitives as XML documents and decode the data from the receiving "software" service primitives. FDT uses this method for all interfaces. It can be seen as one of the main interface methods for vertical communication using XML technology.

By means of FDT specifications and the corresponding interface technology, the user (engineering system manufacturer) will be able to handle devices and their integration into engineering tools and other frameworks in a consistent manner.

1.3 Fieldbus Profiles

Profiles are functional agreements between field device manufacturers to provide interoperable device functions of a certain device class to other field devices, controllers or DCS, SCADA, and engineering systems. Device classes have common functional kernels accompanied by the variables and parameters. These common set of variables, parameters and functions are called profiles (Figure 19). Profiles reduce the degrees of freedom using the variety of the communication system and the choice of application functions of the device classes.

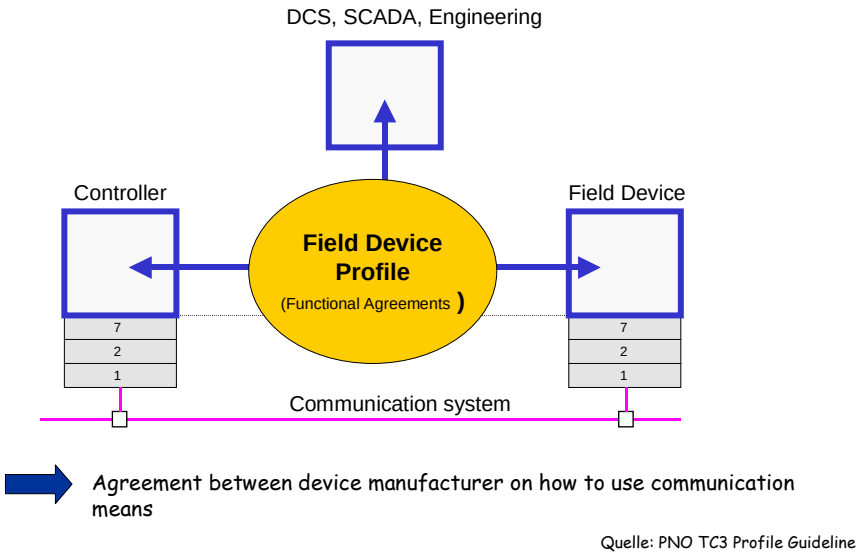


Figure 19: Profile in the automation architecture

There are certain degrees of compatibility and according degrees of co-operation between profile based devices (compatibility levels [19]). Compatibility levels are applicable for various roles of a device e.g.

control, diagnosis, parameterisation/configuration and even applicable to subsets of its functionality. This means that one device can have different compatibility levels regarding different interfaces to the system. The levels are dependent on well-defined communication and application device features (Figure 20).

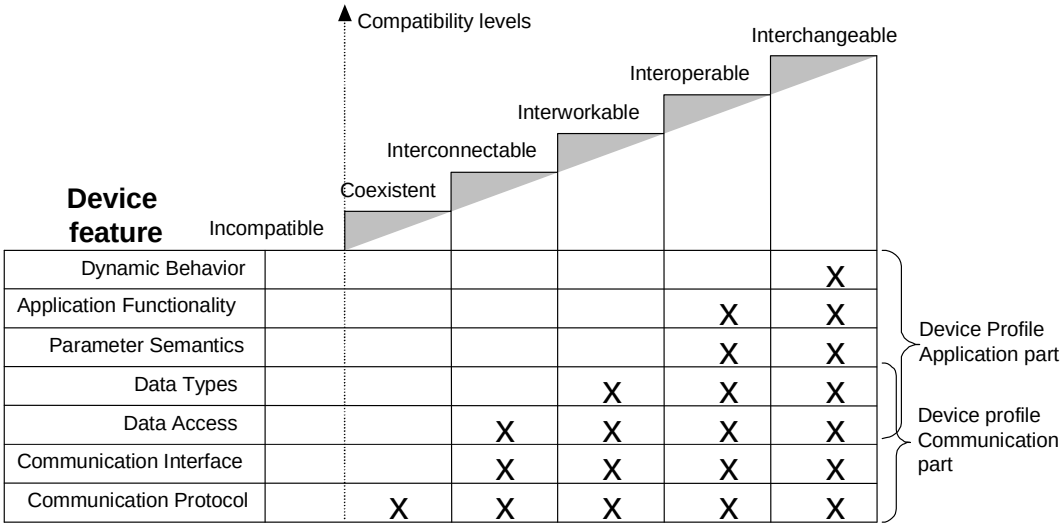


Figure 20: Levels of functional compatibility

The device features are either related to the communication system as it is specified in the according standard (e.g. protocols, service interfaces, data access, data types specified in IEC 61185 and IEC 61784) or related to the device application such as data types and semantic of the parameters, application functions and dynamic behaviour of the application. Profiles usually provide a mixture of compatibility levels regarding parts of the profile and their different users. For example the main measurement value of a device is defined very precisely regarding data type, semantic including dynamic behaviour so devices are interchangeable for this measurement value. The same profile may skip to specify parameters which are used in the function chain from the electrical signal at the process attachment to the measurement value. Then devices are not fully interoperable regarding their parameterisation.

The main benefits of profiles are:

- The state-of-the-art functionality of device classes are specified including their parameter semantic. This makes it possible that human device users as well as tools find the same functions and parameters with the same names and behaviour in devices from different manufacturers.
- It is possible to provide communication feature lists (eg. GSD), EDDs, DTMs and proxies for device classes with profiles.

1.4 Model for Engineering & Instrumentation

The handling of the life cycle of distributed control systems (DCS) is a complex process which can only be done using sophisticated tools (hardware, software). Here it is very important to design a non-interrupted life cycle, i.e. to achieve an information transfer from one step to another step without losing information, and to ensure a single-source principle while putting information into the system. This information is used in each step to create a special view for the user. This can not be done using paper documents for storing and transporting information. However, the usage of data base management systems is not sufficient as long as open technologies are not applied. It is necessary to use databases and communication systems which follow a common agreed transfer syntax and standardized information models which ensure the meaning of the information (semantics). Basically, a connection between all tools in a DCS must be created [5].

There are several possibilities to classify the life cycle/the engineering of DCS [1]. Such classification can be done using:

- the hierarchy of the control functions
- their timing sequence
- their logical dependencies

All life cycle phases which are connected to field devices should be united under the word "instrumentation" and described as use cases. Instrumentation is defined as follows [21]:

> Instrumentation comprises of all activities within the life cycle of the distributed control system where handling of the field devices (logically or physically) is necessary.

Therefore, instrumentation can be considered as the intersection of the life cycle of the distributed control system and the field device. Figure 21 shows the instrumentation steps as a UML use case diagram. There is only one actor, who is not described in detail.

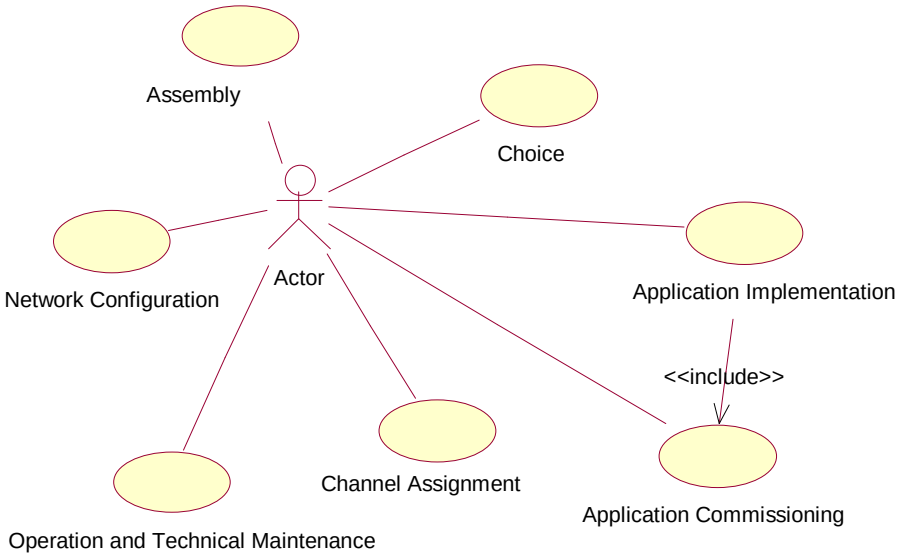


Figure 21: Use case Diagram Instrumentation

1.4.1 Device Model

Field devices are linked both with process, via input /output hardware/ software, and with other devices via communication controllers/ transmission media. The centre of our attention is on the field device as the computational power is increasing rapidly. Thus, the applications are run more and more on these devices and the application processes are becoming more and more distributed. We have to solve the problem of configuring and parameterization of these field devices during the operation for real-time data processing purposes, diagnosis, parameter tuning etc. Therefore, there is a need to model such field devices [5] [6]. A field device can be characterized by:

- internal data management (process I/O image, communication parameters, application parameters)
- process interface
- information processing (e. g. Function Blocks)
- communication interface (Fieldbus, Ethernet-TCP/IP etc.)
- (optional) man/ machine interface (local display, buttons, switches, LEDs)

- (optional) persistent memory and others.

1.4.2 Package structure

We can define a device model as shown in Figure 22 (represented by UML packages) which supports the data exchange between instrumentation steps. This is a very abstract presentation of a device model. All elements of this model are pre-fixed DI (Device Instrumentation). Only the result of the modeling process has been presented, however not the process to reach this result.

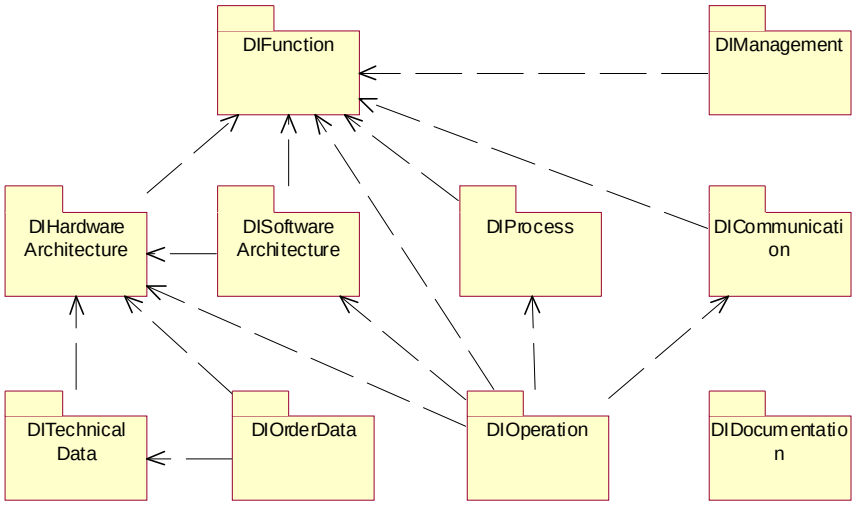


Figure 22: Device model related to the instrumentation

The most important packages (for developers and users) are `DIFunction`, `DIHardwareArchitecture`, `DISoftwareArchitecture`, `DIProcess`, `DICommunication`, `DIManagement` and `DIOperation`. The packages depicted in Figure 21 contain the detailed model represented by UML class diagrams modeling the different views on a device.

The packages which are not described in detail in this paper are briefly introduced as follows:

- `DIHardwareArchitecture`

  hardware modularity of the device with slots, modules, slot rules, etc.

- `DISoftwareArchitecture`

  software modularity of the device with hierarchies of function blocks, etc.

- `DIProcess`

  interface to the physical process

- `DICommunication`

  integration within communication networks

- `DIOperation`

  operation of the device with menus, operation workflows, etc.

- `DIDocumentation`

  documentation as textual and graphical transformation of device features

- `DIOrderData`

  ordering data, calculation of offers, etc.

- `DITechnicalData`

  other technical data as e.g. environmental constraints, mechanical, etc.

1.4.3 Structure of DIFunction

The package `DIFunction` models the functional view on the device. This mirrors two very common views on field devices (device is a list of parameters/variables, device is a (sub) network of function blocks). Figure 22 depicts the class diagram of the package `DIFunction` (presentation as UML class diagram, detail of Figure 22).

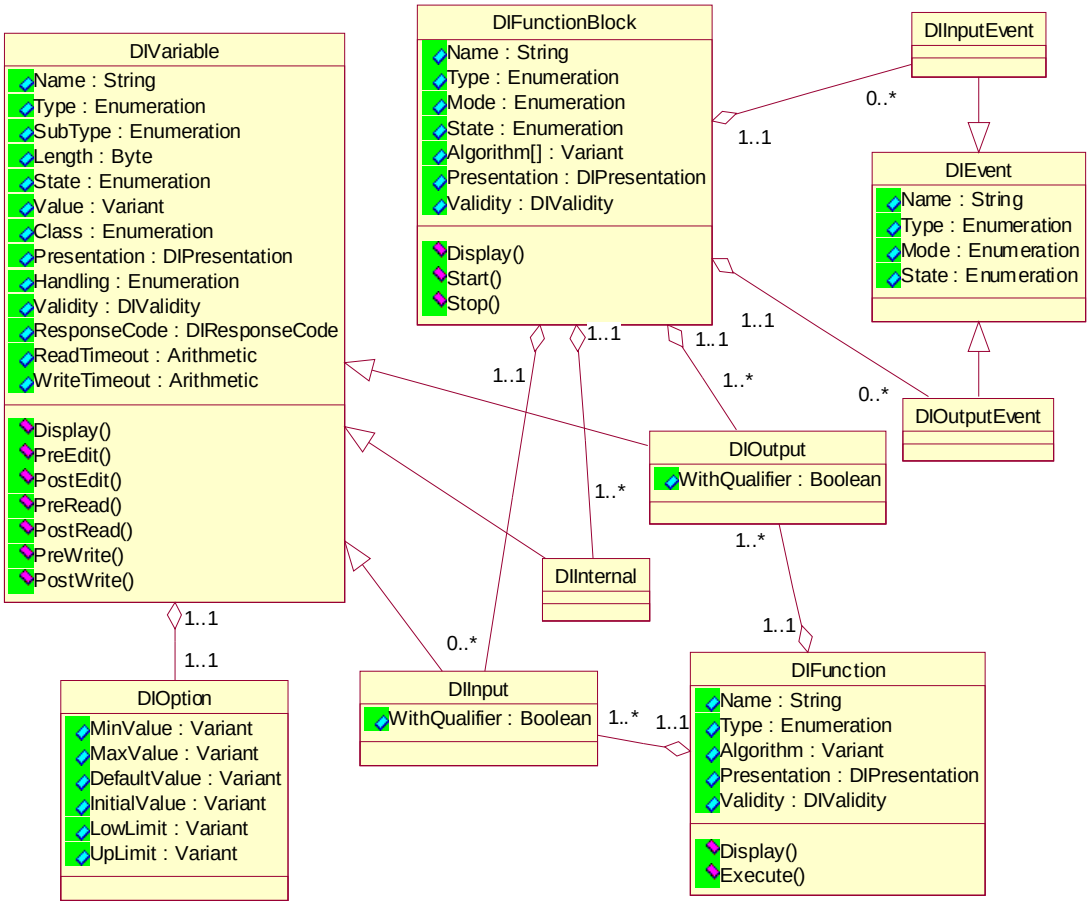


Figure 23: Package DIFunction

`DIFunction` basically follows the function block model of IEC61499. The class attributes/methods reflect the most important features of the IEC61499 model. The classes are characterised as follows:

- `DIFunctionBlock`

  Function block, can be hierarchically structured, identified by its name. The instance is of a certain type (e.g. PID). The current state of the internal state machine is available, and covers the internal execution control and takes the required mode into account. The algorithm which has to be executed can be programmed e.g. in one of the IEC61131 languages. The function block processes input variables using internal variables (stored between consecutive executions of the function block) and algorithms chosen by the internal execution control into output variables. The basic presentation elements are available. The validity of the function block can be controlled.

The function block is integrated into the execution control of the resource. Additionally, it contains its own internal execution control, which can process incoming events and create outgoing events. Therefore, event-driven (IEC61499) and time-driven (IEC61131) systems as well as combinations of both can be described.

To avoid a too complicated presentation, relations to item arrays and item (variable) collections (comp. package `DISoftwareArchitecture`) are not shown here. Function blocks are enabled by this way to use structured variables.

- `DIEvent`

  Event, identified by its name. The instance is of a certain type. The current state of the internal state machine and the mode are available. Events influence the execution control of function blocks.

- `DIFunction`

  Function, can be hierarchically structured, identified by its name. The instance is of a certain type (e.g. Addition). The algorithm which has to be executed can be programmed e.g. in one of the IEC61131 languages. The function processes input variables into output variables without storing anything between two consecutive executions. The basic presentation elements are available. The validity of the function can be controlled.

  To avoid a too complicated presentation, relations to item arrays and item (variable) collections (comp. package `DISoftwareArchitecture`) are not shown here. Functions are enabled by this way to use structured variables.

- `DIVariable`

  Variable, identified by its name. The instance is of a certain type (e.g. arithmetic) and subtype (e.g. integer) and is further specified by a length and by assignment to a class. The current value and state (based on an internal calculation algorithm) as well as the basic presentation elements are available. Furthermore, the handling (read, write), possible error messages, time-out times for reading and writing as well as some other options which depend on the type/subtype are described. The validity of the variable can be controlled.

Programming as part of the application implementation is a phase in the life cycle, in which e.g. function block types, function types and variable types are managed in libraries by certain tools. These types are provided by the libraries and then used as instances. This is not explicitly modelled in the class diagram, i.e. the class `DIFunctionBlock` contains all attributes of the combined classes `DIFunctionBlockInstance` and `DIFunctionBlockType`.

1.4.4 Structure of DIManagement

The package `DIManagement` models the management units of the device. Figure 23 depicts the class diagram of the package `DIManagement` (presentation as UML class diagram, detail of Figure 22).

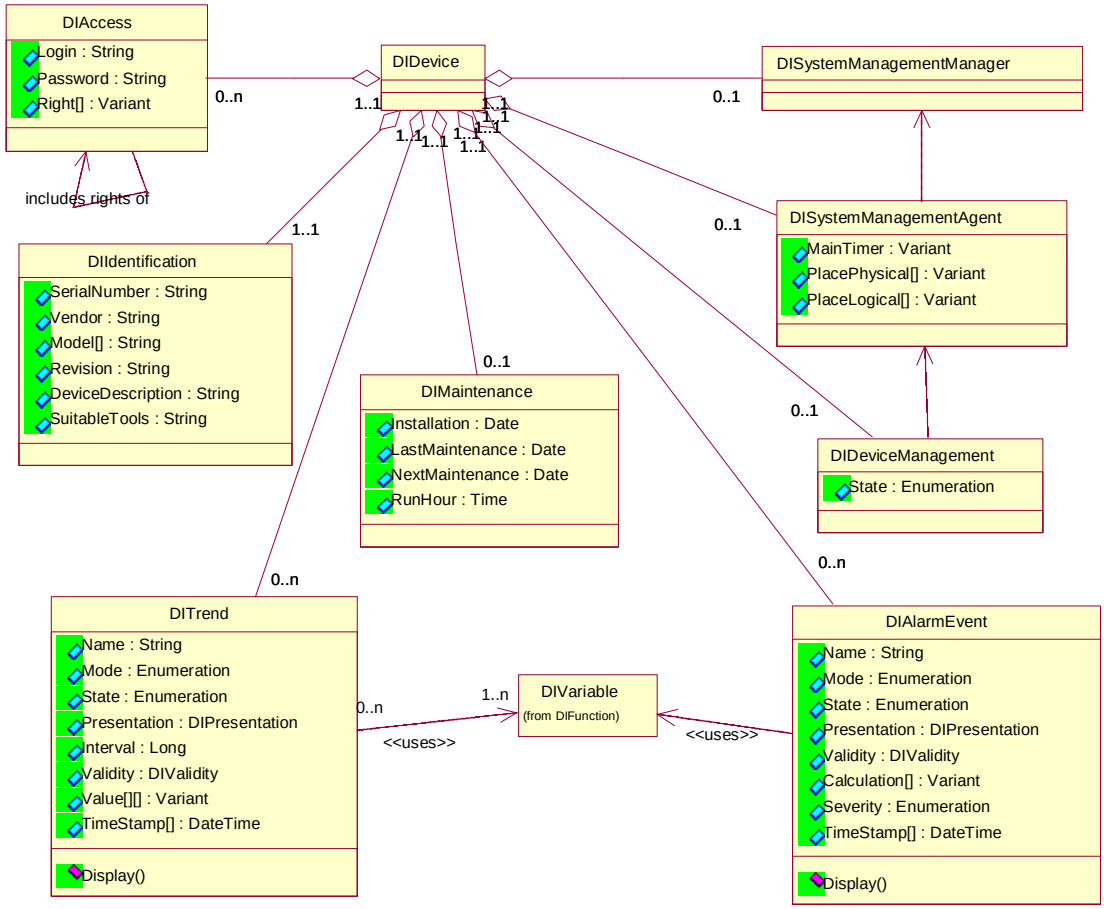


Figure 24: Package DIManagement

The most important features related to its management are modeled. The classes are characterized as follows:

- `DIDevice`

  Main class of the model, instantiated exactly one time, all other classes can be reached from here. DIDevice does not contain its own attributes.

- `DIIdentification`

  Used for unambiguous identification of the device. Contains serial number, vendor, model name (multi-lingual) and revision index. The description of the device is available as device description or object dictionary. Additionally, there is an indication to tools (and data formats) which can handle the device (backward reference). DIIdentification is also used for modules (Package DIHardwareArchitecture).

- `DIAccess`

  Access restriction for the device, can be organized in several levels (e.g. operator, maintenance person, specialist). One level can include the access rights of other levels. The access restriction relates to rights which e.g. concern the execution of functions or the access to variables.

- `DIDeviceManagement`

  The device management comprises of the main state machine (state). The device management acts as agent in a manager / agent relationship with the system management of the device as well as manager with other agents within the device (e.g. execution control within the modules).

- `DISystemManagementAgent`

  The system management agent (device part as agent in a manager / agent relationship to the system management of the overall distributed control system) contains e.g. time synchronization, influences the device management (as manager in a manager / agent relationship). Additionally, the system management agent contains the location information of the device (e.g. the place where the device is located as absolute or relative value, this is used for Application Implementation).

- `DISystemManagementManager`

  The system management manager is the device part as manager in a manager / agent relationship to other devices. It is possible that there are devices which additionally take over the system management of the whole distributed control system, i.e. which are managers in this respect to other devices.

- `DIMaintenance`

  Maintenance, comprises of time stamps for installation, last maintenance, next planned maintenance and an operation hour counter.

- `DITrend`

  Trend, identified by its name. The current state is available. The mode is used for controlling the state. The basic presentation elements are available. At determined points in time the value and the state of all variables (or other objects / attributes) related to the trend are stored. The validity of the trend can be controlled.

- `DIAlarmEvent`

  Alarm event, identified by its name. The current state is available. The mode is used for controlling the state. The basic presentation elements are available. Events created by variables (or other objects / attributes) related to the alarm event are acquired and valued. The validity of the alarm event can be controlled.

[27] contains all details of all other class diagrams needed for modeling the semantics of field devices. This model has to be described by description languages to generate the basic information for an uninterrupted tool chain.

1.4.5 Description and Realization Opportunities

The device model can be implemented (realized) in several ways. It is possible to derive the FDT/DTM structure and interfaces, EDD, field device proxies, function blocks and other technologies from this device model. For this paper we chose EDD and a XML based language. For the computable description of device parameters for automation systems components, the so called Electronic Device Description Language (EDDL) has been specified [16] and [26]. EDD is used to describe the configuration and operational behavior of a device (see 3.2).

1.4.5.1 Overall Example using EDD

EDD is based on the ASCII standard. XML (Extended Markup Language) could be a promising approach for the future, especially because of its use in other areas. Both approaches contain definitions for the exchange of device descriptions using files. These definitions are not given here, however a small example is used to show that different realizations can and must be based on the same solid foundation - the device model.

The example comprises of a variable (package `DIFunction`), which is described by name, data type, label and help.

The class `DIVariableExample` as shown in the following figure is the starting point.

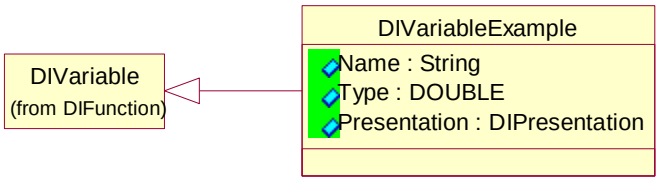


Figure 25: Class DIVariableExample

The realization as Electronic Device Description Language is described using language production rules shown in Figure 26.

```
variable
  = 'VARIABLE' Identifier '{' variable_attribute_list '}'

variable_attribute_list
  = variable_attribute_listR

variable_attribute_listR
  = variable_attribute
  = variable_attribute_listR variable_attribute

variable_attribute
  = help
  = label
  = type
```

Figure 26: Language production rules (EDD)

A sentence created according to these rules may look like Figure 27.

```
VARIABLE Temperature
{
  LABEL       "Temperatur";
  TYPE        DOUBLE;
  HELP        "Temperatur, gemessen am Sensor";
}
```

Figure 27: A variable definition (EDD)

Using this definition of a variable, a commissioning tool provides the following human machine interface:

Figure 28: Human machine interface showing a variable

The unit (Kelvin) is not supported by the example model. The help text is not shown, the name of the variable is not used. Based on the data type, the value provided by the device is shown.

1.4.5.2 The XML approach

The eXtensible Markup Language XML [3] expands the description language HTML with user-defined tags, data types and structures. In addition, a clear separation between the data descriptions, the data itself, and

its representation in a browser has been introduced. Furthermore, declaring syntactical and semantic information in a separate file (Document Type Definition, DTD), allows re-using the description structure in different contexts. This provides a number of benefits when using the same XML description file for different tasks. Different views can be implemented on top of the same data. The description can be hierarchically organized. Depending on the functions to be performed, the XML data can be filtered and associated to software components (controls, Java beans, etc.). The selection of the necessary information and the definition of their presentation details can be performed by means of scripts and style sheets. The style sheets are part of the development of XML [2]. In most cases they are implemented using the extensible Style Language (XSL). The XML file, the scripts and the different style sheets can be used to generate HTML pages, special text files, and binary files (components, applets) necessary to build the certain functions of the software tools. The distribution of the generated HTML pages and associated software components is done following the concepts used in an Internet environment. The major benefit of this solution is a unique, reusable description with an excellent consistency and reduced efforts of the description process.

For the realization using the XML approach, the specification of a schema is necessary. Figure 29 shows part of it describing the element variable.

```
<ElementType name="DIVariable" content="mixed" model="closed">
  <attribute type="Name" required="yes"/>
  <element type="operation:DIPresentation" minOccurs="1" maxOccurs="1"/>
  <element type="Type" minOccurs="1" maxOccurs="1"/>
</ElementType>
```

Figure 29: Schema specification

An instance of this schema may look like the following:

```
<DIVariable
  Name="Temperature"
  <operation:DIPresentation>
    <operation:Label
      String="Temperatur">
      </operation:Label>
      <operation:Help
        String="Temperatur, gemessen am Sensor ">
      </operation:Help>
    </operation:DIPresentation>
    <Type>
      <Arithmetic>
        <Double
        </Double>
      </Arithmetic>
    </Type>
</DIVariable>
```

Figure 30: A variable definition (XML)

A standard web browser creates the following interface:

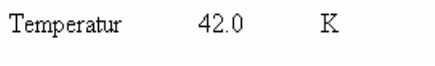


Figure 31: Web browser showing a variable

The unit (Kelvin) is not supported by the example model. The help text is not visible, the name of the variable is not used. Based on the data type, the value provided by the device is shown.

The presentation of the small example underlines the objectives targeted by the modeling approach. If the internal structures of different realizations are similar, i.e. follow the same field device model, then it is possible to build translators from one realization to another and to secure investments already done. Similar presentations with the same contents provide the opportunity to simplify training and education through previous recognition.

Device Descriptions are necessary for the integration of intelligent field devices in commissioning tools, maintenance tools, engineering systems or MES / ERP systems. Device Descriptions comprise of device models and presentations based on the models (e.g. ASCII files). The XML concept may help to enlarge the application scope of Device Description because it is becoming one of the basic technologies of the fast growing internet and various e-engineering activities. At present, we can observe a transition to XML based Device Descriptions which is characterized by the following issues:

- XML and ASCII based Device Description have to use the same device model to keep the semantics already developed by automation industry
- new application functions supporting different phases of the life cycle of an automation system have to be defined using different views on the Device Description.
- consequently, a new chain of tools for creating and using Device Descriptions has to be developed.

A necessary precondition is the international standardization in this area, which is currently done in the European standardization CENELEC and the IEC.

1.5 Usage of proxy

1.5.1 General

Another approach for the integration of different devices into an automation engineering system is the usage of a proxy. The proxy follows the *proxy pattern*, a well known software design pattern. A client who wants to communicate with a device, talks not directly with the device rather with its representative – the proxy. The advantages of such a placeholder are multifaceted. It offers efficient efficiency, easier access or additional protection against unauthorized access.

Figure 32 shows the UML class diagram of the design pattern graphically. The Device offers the services, in the area of automation first of all data. Legacy devices offer a data interface, not a service oriented interface. The Client is interested on the data of the Device or would like to use its services. Therefore it invokes the functionality of the original in an indirect way by means of the Proxy. Therefore the Proxy offers the same interface as the original (AbstractOriginal). The proxy has a one-to-one relationship to the device in order to communicate with the device directly.

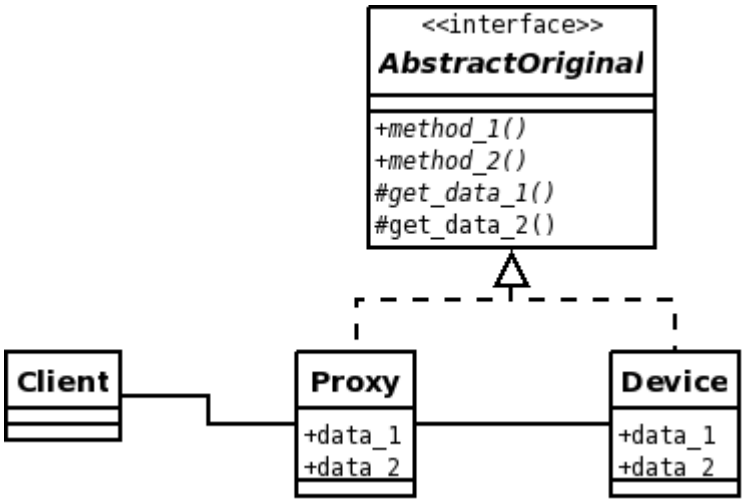


Figure 32: Proxy pattern (adapted from [4])

Adapted to the area of automation, it is more efficient to install a central proxy, which provides access to one or several devices. So, the specific communication between the proxy and the devices is out of the scope of the client.

There are several variants of the proxy design pattern. Interesting for automation area are the *remote proxy* and the *cache proxy* in order to get data from a whole plant or be more effective for the relatively slow fieldbus communication.

Very popular inside the last decade are proxies, based on the OPC Data Access specification.

1.5.2 Overview about available OPC specifications

Originally OPC is the abbreviation for OLE for Process Control. OPC should allow client applications access to plant floor data in a consistent manner. Especially the system and device integration in heterogeneous environments should become simpler.

OPC is one very popular set of specifications in order to standardize the communication of process control data. Based on the available specifications an infrastructure will be provided. It is the task of the vendors to provide specific devices / software tools, complying with the OPC specifications. Additionally, OPC should reduce implementation time and costs by offering a true interoperable and scalable system

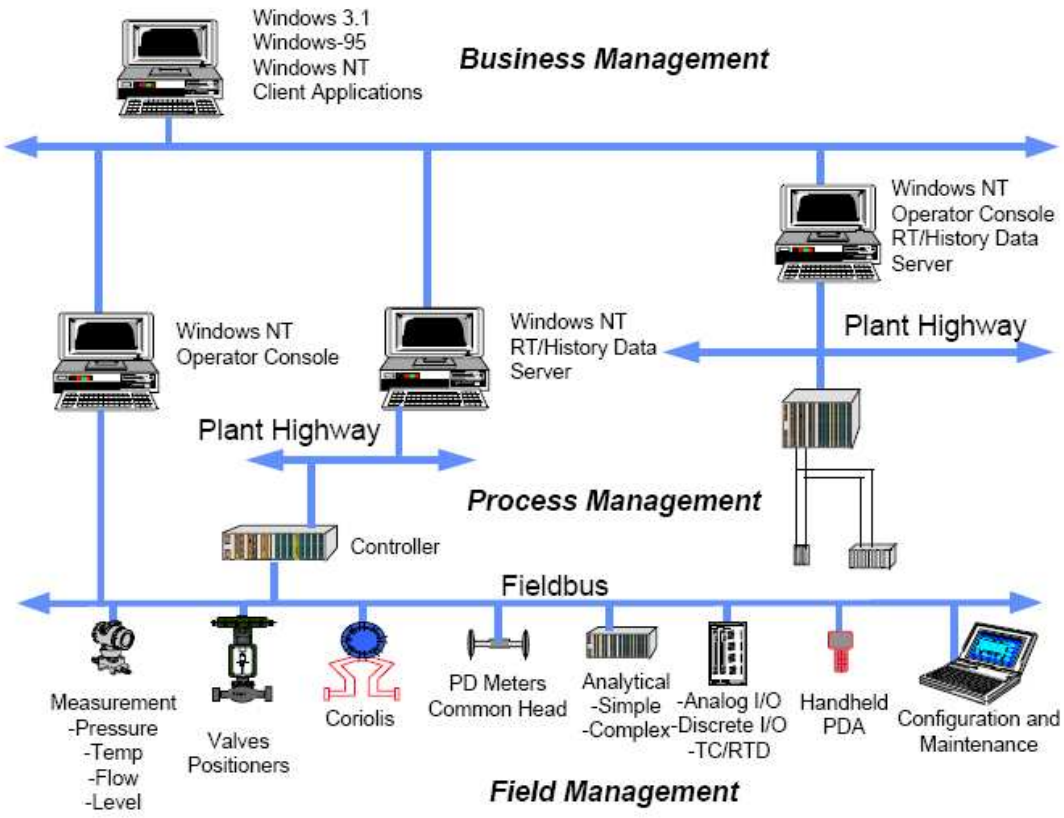


Figure 33: Process Control Information Architecture [22]

Inside Figure 33 the *Field Management* is responsible for providing data received from or sending to the smart field devices. The transmitted data represent information about the device itself (e.g. state, manufacturer, revison) or data about the process (e.g. control or status information). All this information must be presented to the user, and any applications using it, in a consistent manner.

Typical systems inside the *Process Management* are SCADA or DCS systems, in order to monitor or control manufacturing processes.

In additional benefit can be achieved inside the Business Management by means of a consistent view about all sub layered information.

OPC offers an efficient vertical path through the complete plant hierarchy from the shop floor up to the business systems. OPC provides an architecture for an efficient data access, independent of its data type.

In the past, vendors would capture this data in their own applications by defining own, proprietary interfaces. So for each systems a different device driver or proxy system would be necessary with the disadvantage, that only the own, know data formats could be processed. By using the OPC set of standards, data can be passed from any data source to any OPC compliant application.

OPC uses Microsoft's COM and DCOM technology to enable applications to exchange data on one or more computers using a client / server architecture. OPC defines a common set of interfaces. So applications retrieve data in exactly the same format regardless of whether the data source is a PLC, DCS or anything else. As a result, OPC is as an out-of-the-box, plug and play communication solution [20].

Available OPC specifications:

- OPC Data Access, or OPC DA, provides access to real time process data. Using OPC DA, one can ask the OPC server for the most recent values of flows, pressures, levels, temperatures, densities, and more.

- OPC Historical Data Access, or OPC HDA, is used to retrieve and analyze historical process data, which is typically stored in a Process Data Archiver, database, or RTU.
- OPC Alarms and Events, or OPC A&E, is used to exchange process alarms and events. Operations personnel can use OPC A&E to notify them of alarms and obtain a sequence of events.
- OPC Data eXchange, or OPC DX, defines how OPC servers exchange data with other OPC servers.
- OPC Extensible Markup Language, better known as OPC XML, encapsulates process control data making it available across all operating systems.
- There are also other specifications such as OPC Batch and OPC Security.

As a next step in order to reduce the operating system dependencies, the OPC UA (Unified Architecture) is emerging. OPC UA extends communication to production planning and ERP systems. OPC UA is based on Web Services, which is a nonproprietary communication protocol. With XML and Web services as machine to machine communication, OPC becomes a service-oriented architecture.

1.5.3 Approach for representation of devices inside an OPC server

The OPC specifications are very universal, therefore concrete predefinitions, how an OPC server has to present a specific device, are not included. This gap will be filled normally by the real implementations of OPC servers of different vendors. A disadvantage of this approach is non unique representation of the information across the vendors. The DRIVECOM user group [7] developed a specification called 'DriveServer' addressing this topic [8]. This specification defines the communication interface for accessing drives.

From the user's point of view a device (inside the DRIVECOM a drive is a synonymous to device) is something completely different than it is from the communication's point of view. Users work with parameters, which shall be detectable by names. The DriveServer specification is also based on the OPC interface standard, so interfaces in process automation of the individual adaptation to controllers and hardware is not longer necessary. It is comparable to a layer between a user program with OPC client interface and the communication media (e.g. fieldbus). The connection with the communication media can either be implemented by the vendor in the DriveServer or a communication OPC server. The DriveServer encapsulates device features according to their functions. All features are accessed via a functional interface. Thus the functionality of the devices, their description and access mechanism remains unchanged. Access is made via standard means, for instance, not parameters and their content but the access to these parameters are defined by names. The main advantage is that the vendors do not have to change their implementations.

The DriveServer specification is based on a clear separation between communication functionality and DriveServer functionality. By this it is ensured that the great variety of available and reasonable communication media can be used. A DriveServer provides the functionality necessary to access and handle a drive. The DriveServer communicates via an OPC interface. This open architecture ensures the flexibility of fieldbus systems required, see Figure 34.

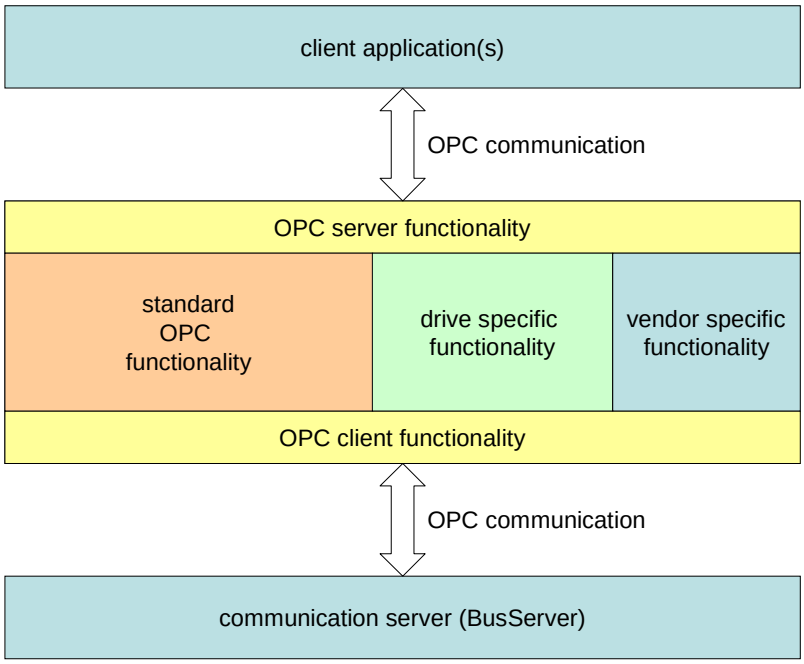


Figure 34: DriveServer Architecture

At first the DriveServer itself is a client of a communication OPC server (also called BusServer) since the communication interfaces usually connect fieldbuses. The BusServer can however be a server of any transfermedium. The DriveServer maps the device-related accesses of the application to fieldbus-related accesses. According to the OPC specification, DriveServer and communications OPC server access use names for accessing. These names can be preconfigured or have a dynamic character, depending on the server implementation. The interface between application, DriveServer and BusServer comprises three phases:

- Device Identification
- Item definition as access specification (configuration)
- I/O operations (transfer)

For interoperability issues, the provided syntax of the items and the semantic of the some identification related items are defined.

An open issue inside the DriveServer specification is the configuration of the DriveServer itself. The server provides the functionality to identify the available drives, offered by the related BusServers, but the configuration depending on the drive types is a vendor specific task. Thus DRIVECOM developed also a device description, based on XML in order to configure the DriveServer. This is normally a business of the vendor, but it could also based by the offered approach, see Figure 35.

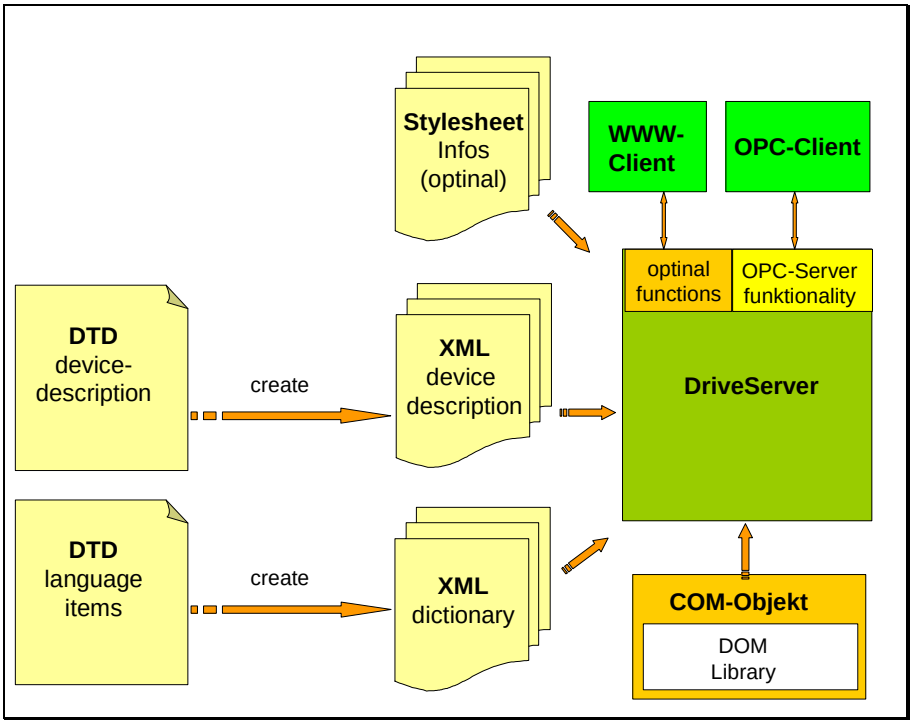


Figure 35: Integration of device description into the DriveServer

The system architecture comprises the language definition in the form of the DTD, the definition of the dictionary in the form of a DTD, numerous device descriptions (XML files) that are loaded onto the DriveServer, and the country-specific language files. The DriveServer can also add layout information in the form of style sheets. The device description contains all the information necessary to describe the drive. For this purpose the descriptions can be divided into standard and device-specific descriptions. Standard descriptions are imported by the actual description and can be specified by groups of manufacturers. Then the manufacturer must only describe the additional parts.

2 Configuration approach for web service enabled devices

2.1 Use cases for web services

The intention of the Service Oriented Architecture is the usage of service, provided by servers. In the context of automation devices, each device has to provide a specific set of such services. These services can be used in order to build loosely coupled systems. The advantage of such loosely coupled systems is the convenience of establishing connections between such services. A disadvantage is very often the lower performance of such loosely coupled system in contrast to tightly coupled systems. This disadvantage is from the point of view of commissioning or maintenance not so relevant.

In order to achieve data from the device, legacy devices provide a data oriented interface. This interface will be used in the form of simple communication services, e.g. *Read* / *Write*. In order to get information from Web Service enabled devices, the paradigm of the device interfaces must be revised. A modification of the paradigm is a tedious mission and should be accepted by a variety of device vendors. Thus this revision process could be done in several consecutive steps, in order to gain experiences in each smaller step.

2.2 Usage of same semantics of communication services

In a first step the same semantics of the communication services could be transformed into the syntax of web services. This means, one could use *Read* / *Write* services for the data transmission and higher level applications have to now the semantic of each data item. This approach is equivalent to the recent state of communication with automation devices.

2.3 Usage of high level services

The next steps are characterized by the transformations of the data oriented interface to providing high level services. One promising approach could be the usage of the semantic of available Function Block oriented device profiles [16]. The device internal structure is normally divided into specific functions, which can also be instantiated in form of blocks. Legacy devices allow only an indirect modification of the offered functions in form of configuration parameters. This adaptation of the device can also be done by a set of offered web services, e.g. `Set_SetPoint(), Get_Config(), Config_PID()`.

Each device offers also its web service by means of device description. In the context of web services such a device description is called WSDL (Web Services Description Language)

3 Maintenance capabilities provided by devices

Maintenance capabilities should be used in order to reduce the downtime of a device or a part / whole automation system. Maintenance issues could be divided into two general parts:

- Corrective maintenance

  This issue is very important and required immediately actions by the maintenance staff in order to repair / replace the device or component inside the automation system.

- Preventive maintenance

  This category could also be subdivided into

  - Condition based

    Inside this category, the device or component should send data describing the detected problem. Normally the device or component is active but is out of it optimal operating point.

  - Time based

    A predefined schedule defines the maintenance interval of a device or component. The duration can either be defined by the device manufacturer or can be provided by the end user plant experiences.

  - Predictive

    Very often heuristics are used for this strategy of maintenance. The continuously measured values must be compared with the heuristics or characteristics of the device or components. The derived maintenance action may be determined by approximation of when an error might occur.

Especially for the preventive maintenance tasks recent devices offer some information about its state and will be used as diagnostic data. The determination, which data is a diagnostic relevant data or not, will either be predefined inside profiles or described inside a device description. The device description solves the detection problem also by providing predefined names of the relevant parameters

4 Acronyms

DCS	Distributed Control System
DTD	Document Type Definition
DTM	Device Type Manager
EDDL	Electronic Device Description Language
EDS	Electronic Data Sheet
ERP	Enterprise Resource Planning

FDT	Field Device Tool
FF	Fieldbus Foundation
MES	Manufacturing Execution System
OLE	Object Linking and Embedding
PLC	Programmable Logic Controller
SCADA	Supervisory Control and Data Acquisition
UML	Unified Modelling Language
WSDL	Web Services Description Language
XML	eXtensible Markup Language

5 References

[1] Alzenauer, R.: Semantic information model for an overall, computer based engineering using the process industry as an example(in German). PhD thesis, Rheinisch Westfälische Technische Hochschule Aachen. 1998.

[2] Boumphrey, F.: Professional Style Sheets for HTML and XML. Wrox Press. 1998.

[3] Bray, T., Paoli, J., Sperberg-McQueen, C. M.; Maler, E.; Yergeau, F.: Extensible Markup Language (XML) 1.0 (Fourth Edition), http://www.w3.org/TR/REC-xml, 2006

[4] Buschmann, F.; Meunier, R.; Rohnert, H.; Sommerlad, P.; Stal, M.: Pattern-Oriented Software Architecture - A System of Patterns, Wiley & Sons, 1996

[5] Diedrich, Ch., Neumann, P.: Field device integration in DCS engineering using a device model, IECON'98, IEEE Conference, Proceedings pp. 164-168, Aachen, 1998

[6] Diedrich, Ch., Wollschlaeger, M., Riedl, M., Simon, R.: Three Component Model for field device integration in control systems. IFAC/FET 2001. Nancy (France), 15.-16.11.2001. Proceedings, 2001

[7] DRIVECOM: DRIVECOM User Group e.V., www.drivecom.org, 2001

[8] DRIVECOM: DriveServer specification, 2001

[9] DRIVECOM: Device Description, 2002

[10] FDT Interface Specification, Version 1.0. PROFIBUS Guideline. PROFIBUS User Organization, September 2000.

[11] FDT Interface Specification, Version 1.2.1, FDT Technology Specifications .http://www.fdt-jig.com/en/10e_down/de-01_spez-b.html, 2005

[12] Foundation Specification, Device Description Language, Fieldbus Foundation, Austin Texas, 1996.

[13] HART: Device Description Language specification: HCF Austin Texas, 1995.

[14] IDA: IDA – the Internet of Automation Technology. White paper V1.0 April 2001. www.ida-group.org, 2001

[15] IEC 61499: Function Blocks for industrial-process measurement and control systems – Part1 and 2. Public Available Specification (PAS) Geneva, 2001.

[16] IEC 61804-2: Specification of FB concept and Electronic Device Description Langauges (EDDL). Committee Draft for Vote (CDV), Geneva, 2003.

[17] Riedl, M.; Naumann, F.: EDD-Entwicklung up to date, Computer&Automation, 01/06, S. 32-35

[18] Diedrich, Ch.; Riedl, M.: Gestalterische und funktionale Gewinne durch EDD-Erweiterungen, atp - Automatisierungstechnische Praxis 48 (2006) Heft 2, S. 24-27.

[19] IEC: Device Profile Guideline: Device Profile Guideline - Draft for Public Available Specification (PAS), Geneva, 2003.

[20] Matrikon: OPC Tutorial, www.matrikon.com, 2003

[21] Neumann, P.; Simon R.; Diedrich, Ch.; Riedl, M. (2001). Field Device Integration. 8th IEEE International Conference on Emerging Technologies and Factory Automation. ETFA 2001, Proceedings pp.63-68, Antibes, 2001

[22] OPC Foundation: OLE for Process Control, OPC Overview, Version 1.0, 1998

[23] PNO: Vol. 1: GSD Specification; Specification for PROFIBUS Device Description and Device Integration, PROFIBUS User Organisation, 2005

[24] PNO: Electronic Device Description, Version 1.2. PROFIBUS Guideline. PROFIBUS User Organisation, 2005.

[25] PROFInet: PROFInet – more then just Ethernet. Broschure www.profibus.com, 2002.

[26] Simon R., Demartini, C. (1999b). Electronic Device Description, Fet'99, 23./24.09.1999, Magdeburg, Proc. of FET'99, pp.429-436, ISBN 3-211- 83394-3, Springer Verlag Wien New York, 1999

[27] Simon, R.: Methods for Field Instrumentation of Distributed Computer Control Systems (in German). PhD Thesis, Otto-von-Guericke University Magdeburg, 2001.

[28] ARC brief: FDT Open Access to Device Intelligence Unlocks Interoperability to Bridge Information Silos, ARC Whitepaper, 2006.

Annex A – Dissemination Levels

PUBLIC	Public
PP	Restricted to other programme participants (including the Commission
RESTRICTED	Restricted to a group specified by the consortium (including the Commission
CONFIDENTIAL	Confidential, only for members of the consortium (including the Commission Services)

Table 1: Dissemination levels for a document