ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta elektrotechnická – Katedra řídící techniky

# DIPLOMOVÁ PRÁCE

## FDT/DTM v průmyslové automatizaci

Jiří Tomášek

květen 2007

# Anotace

Diplomová práce se zabývá návrhem ovladačů pro provozní přístroje umístěné na průmyslové sběrnici PROFIBUS PA podle koncepce Field Device Tool/Device Type Manager (FDT/DTM). Návrh je založen na nástroji dtmManager společnosti M&M Software GmbH, který slouží k tvorbě modulů DTM. K propojení aplikace FDT umístěné na počítačové stanici s průmyslovou sběrnicí je použita komunikační karta CP5611 společnosti Siemens.

V teoretické části diplomové práce se nachází seznámení s principy popisu provozních přístrojů na základě programových komponent, stručný popis průmyslové sběrnice PROFIBUS PA a popis nástroje dtmManager.

Praktická část popisuje návrh komunikačního modulu a ovladače pro provozní přístroj. Jako tento přístroj bylo vybráno teplotní čidlo TF-211 společnosti ABB Automation GmbH.

# Annotation

Diploma thesis is concerned with project of drivers for field devices connected to process fieldbus PROFIBUS PA, dealing with conception of Field Device Tool/Device Type Manager (FDT/DTM). Project is based on dtmManager by M&M Software GmbH, which is used for creation of DTM modules. Communication card CP5611 by Siemens was used for connecting FDT application, placed on PC station, to process fieldbus.

In the theoretical part of my diploma thesis, there takes place an introduction to principles of description of field devices, based on programme components, a brief description of process fieldbus PROFIBUS PA and characterization of dtmManager.

Practical part describes the project of the communication module and the device driver for the temperature sensor TF-211 by ABB Automation GmbH.

# Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.
Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne …… ……
podpis

## Poděkování

Tímto bych chtěl poděkovat svým rodičům a rodině, kteří mi poskytli zázemí a podporu po dobu tvorby mé diplomové práce i mého studia, mým přátelům, jež mi byli oporou v těžkých chvílích, a v neposlední řadě i Ing. Pavlu Burgetovi za pomoc při tvorbě diplomové práce.

Zvláštní dík pak patří Andreasu Trefzerovi a Volkeru Herbstovi z M&M Software, jejichž neocenitelné rady přispěly k úspěšnému dokončení této diplomové práce.

# Obsah

1. Úvod ... 10
2. Nástroje pro integraci přístrojů do řídícího systému ... 11
2.1. Úvod ... 11
2.2. Standard General Station Data (GSD) ... 12
2.3. Standard Electronic Device Description (EDD) ... 12
2.4. Standard Field Device Tool/Device Type Manager (FDT/DTM) ... 13
3. Specifikace standardu FDT/DTM ... 14
3.1. Vznik a cíle standardu FDT/DTM ... 14
3.2. Technologické pojetí ... 14
3.3. Technologické řešení ... 16
4. Základy FDT ... 18
4.1. Základní myšlenka FDT ... 18
4.2. Základní architektura FDT ... 18
4.3. Základní objekty a rozhraní FDT ... 19
4.3.1. Device Type Manager (DTM) ... 19
4.3.2. Block Type Manager (BTM) ... 20
4.3.3. Rámcová aplikace FDT ... 21
4.4. Komunikace pomocí XML ... 22
4.5. Uchovávání trvalých dat v FDT ... 23
4.6. Přístup na sběrnici ... 24
4.7. Operační fáze a přístupová práva ... 24
5. Základní koncepty FDT ... 26
5.1. Stavové schéma DTM ... 26
5.2. Objektový model FDT ... 27
5.2.1. Objekt rámcové aplikace (FrameApplication) ... 28
5.2.2. Objekt projektu (Project) ... 28
5.2.3. Objekt DTM ... 28
5.2.4. Objekt BTM ... 28
5.2.5. Objekt kanálu FdtChannel ... 29
5.2.6. Objekt prezentace (Presentation) ... 29
5.2.7. Objekt DcsChannel ... 29

5.2.8. Asociační vazby v modelu ... 29
5.3. Rozhraní pro DTM, BTM a rámcovou aplikaci ... 30
5.3.1. Rozhraní pro DTM ... 32
5.3.2. Rozhraní pro rámcovou aplikaci ... 34
5.3.3. Rozhraní pro BTM ... 35
5.3.4. Rozhraní pro FDT Channel ... 37
5.4. XML schémata FDT ... 39
5.4.1. XML v FDT ... 39
5.4.2. Schéma FDTDataTypesSchema ... 39
5.4.3. Schéma FDTApplicationIdSchema ... 39
5.4.4. Schéma FDTUserInformationSchema ... 39
5.4.5. Schéma DTMInformationSchema ... 40
5.4.6. Schéma DTMParameterSchema ... 41
5.4.7. Schéma DTMFunctionsSchema ... 41
5.4.8. Další XML schémata ... 42
6. PROFIBUS ... 43
6.1. Úvod ... 43
6.2. Rozdělení PROFIBUS ... 43
6.3. PROFIBUS FMS ... 43
6.4. PROFIBUS DP ... 43
6.5. PROFIBUS PA ... 44
6.6. Profily PROFIBUS PA ... 45
6.6.1. Úvod ... 45
6.6.2. Rozdělení profilu PROFIBUS PA ... 45
6.6.3. Struktura profilu PROFIBUS PA ... 46
6.6.4. Správce zařízení a adresář objektů ... 47
6.6.5. Parametry bloků ... 49
7. Vývoj softwarových modulů DTM ... 52
7.1. Úvod ... 52
7.2. Prostředky pro vývoj DTM ... 52
7.2.1. DTMCreator Base (iFak System GmbH) ... 52
7.2.2. dtmManager 2.1 (M&M Software GmbH) ... 53
7.2.3. Volba vývojového nástroje ... 54

7.3. Popis vývojového nástroje dtmManager 2.1 ... 54
7.3.1. Statické a dynamické knihovny ... 54
7.3.2. Hlavičkové soubory ... 55
7.3.3. Soubory se šablonami ... 56
7.3.4. Soubory pro tvorbu GUI ... 56
7.3.5. Základní architektura projektu postaveném na DTMBaseLibrary ... 57
7.3.6. Koncepce callbacků ... 58
7.3.7. Objekt obsluhy funkcí - Function Manager ... 59
7.3.8. Objekt obsluhy uživatelského rozhraní - UI Manager ... 60
7.3.9. Objekt obsluhy perzistentních dat – Persistence Manager ... 61
7.3.10. Obsluha typu zařízení ... 62
7.3.11. Služby rámcové aplikace ... 63
7.3.12. Koncepce rozhraní DTM status ... 63
7.4. Grafického rozhraní v dtmManageru ... 64
7.4.1. Tvorba grafického rozhraní ... 64
7.4.2. Grafické oblasti ... 64
7.4.3. Grafické komponenty ... 67
7.4.4. Validace vstupních dat ... 67
8. DTM modul CommDTM ... 68
8.1. Úvod ... 68
8.2. Architektura CommDTM ... 68
8.3. ATL COM část CommDTM ... 70
8.3.1. Třída CCommDTMCore ... 70
8.4. Výkonné jádro CommDTM ... 71
8.4.1. Třída CCommDTMCoreImpl ... 71
8.4.2. Třída CCommJobHandler ... 72
8.4.3. Třídy typu CommJobs ... 72
8.4.4. Třída CDPV1Channel ... 73
8.5. Obsluha komunikační karty ... 73
8.5.1. Komunikační karta CP-5611 ... 73
8.5.2. Okno zpráv ... 75
8.5.3. Instance karty CP-5611 ... 75
8.5.4. Fronta zpráv ... 77

8.6. Grafické rozhraní pro CommDTM ... 77
9. DTM modul DeviceDTM ... 79
9.1. Úvod ... 79
9.2. Architektura DeviceDTM ... 79
9.3. ATL COM část DeviceDTM ... 80
9.4. Výkonné jádro DeviceDTM ... 81
9.5. Uživatelské rozhraní pro TF-211 ... 85
9.5.1. Architektura GUI pro TF-211 ... 85
9.5.2. Popis grafického prostředí ... 86
9.5.3. Reakce na události ... 88
9.5.4. Čtení ze zařízení ... 89
9.5.5. Zápis do zařízení ... 90
9.6. Uživatelské rozhraní pro obecný teploměr ... 90
10. Rámcová aplikace a moduly DTM ... 91
10.1. Volba rámcové aplikace ... 91
10.2. Práce s rámcovou aplikací ... 91
10.2.1. Device Catalogue ... 91
10.2.2. Network View ... 91
10.2.3. Monitor View ... 92
10.3. Práce s moduly DTM ... 92
10.3.1. Přenositelnost ... 92
10.3.2. Spuštění DTM ... 93
11. Závěr ... 94
12. Použitá literatura ... 96
13. Seznam příloh ... 97

# Seznam obrázků

Obrázek 1 - Vývoj nástrojů pro popis zařízení ... 11
Obrázek 2 - Paralela mezi strukturou softwaru a hardwaru v FDT/DTM ... 15
Obrázek 3 - Architektura FDT ... 18
Obrázek 4 - Rozhraní modulu DTM ... 20
Obrázek 5 - Rozhraní FDT ... 22
Obrázek 6 - Uchovávání a přístup k datům v FDT ... 23
Obrázek 7 - Stavové schéma DTM ... 26
Obrázek 8 - Vztahy mezi objekty v FDT ... 27
Obrázek 9 - Rozhraní pro DTM a rámcovou aplikaci ... 31
Obrázek 10 - Rozhraní pro BTM ... 36
Obrázek 11 - Rozhraní pro FDT Channel ... 38
Obrázek 12 - Příklad použití FDTUserInformationSchema ... 40
Obrázek 13 - Příklad XML dokumentu DTMInformationSchema ... 40
Obrázek 14 - Příklad XML dokumentu DTMFunctionsSchema ... 41
Obrázek 15 - Rozdělení profilu PROFIBUS PA ... 46
Obrázek 16 - Struktura profilu PROFIBUS PA ... 47
Obrázek 17 - Struktura hlavičky adresáře objektů ... 48
Obrázek 18 - Vnitřní struktura v adresáři objektů ... 49
Obrázek 19 - Základní architektura projektu v dtmManageru ... 57
Obrázek 20 - Koncepce callbacků ... 58
Obrázek 21 - Princip funkce Function Manageru ... 59
Obrázek 22 - Použití UI Manageru ... 60
Obrázek 23 - Persistence Manager ... 61
Obrázek 24 - Obsluha typu zařízení ... 62
Obrázek 25 - Koncepce služeb rámcové aplikace ... 63
Obrázek 26 - Koncepce rozhraní DTM status ... 63
Obrázek 27 - Rozložení grafických oblastí ... 65
Obrázek 28 - Příklad XML dokumentu popisujícího grafické prostředí ... 66
Obrázek 29 - Architektura CommDTM ... 69
Obrázek 30 - Použití komunikační karty CP-5611 ... 74
Obrázek 31 - Rámcová aplikace s grafickým rozhraním pro CommDTM ... 78
Obrázek 32 - Architektura DeviceDTM ... 80
Obrázek 33 - Uživatelské rozhraní DeviceDTM ... 86

# Seznam tabulek

Tabulka 1 - Struktura parametru Block Object ... 50
Tabulka 2 - Block Object pro fyzický blok ... 50

# 1. Úvod

S rozvojem moderních provozních přístrojů vznikají i nové požadavky na správu řídících systémů, do kterých jsou tyto přístroje začleněny. Možnosti správy současných řídících systémů se začínají jevit jako nedostatečné a proto výrobci hledají nová technologická řešení, která by dokázala pokrýt náročné požadavky moderních provozních přístrojů a inteligentních členů. Jedním z těchto nových řešení je standard Field Device Tool/Device Type Manager (FDT/DTM).

Standard FDT/DTM je založen na otevřenosti. V jednom systému mohou být bez problému umístěna zařízení od různých výrobců, aniž by docházelo k provozním problémům, jak tomu bylo v minulosti. Řešení FDT/DTM je založeno na programových komponentách DTM. Každá tato komponenta představuje ovladač pro konkrétní zařízení a pracuje na podobném principu jako například ovladač pro tiskárnu. Takovéto řešení ulehčuje správu řídícího systému a značně usnaďnuje začlenění nových provozních prvků do tohoto systému.

Cílem této diplomové práce je navrhnout a vytvořit komunikační driver, který pomocí komunikační karty CP-5611 umístěné na počítačové stanici umožní připojit do aplikace FDT průmyslovou síť PROFIBUS PA. Dalším úkolem je vytvořit driver zařízení pro teplotní čidlo TF-211, které je umístěno v tomto řídícím systému. Kromě tohoto čidla je v systému zapojen ještě ultrazvukový snímač úrovně hladiny Prosonic T.

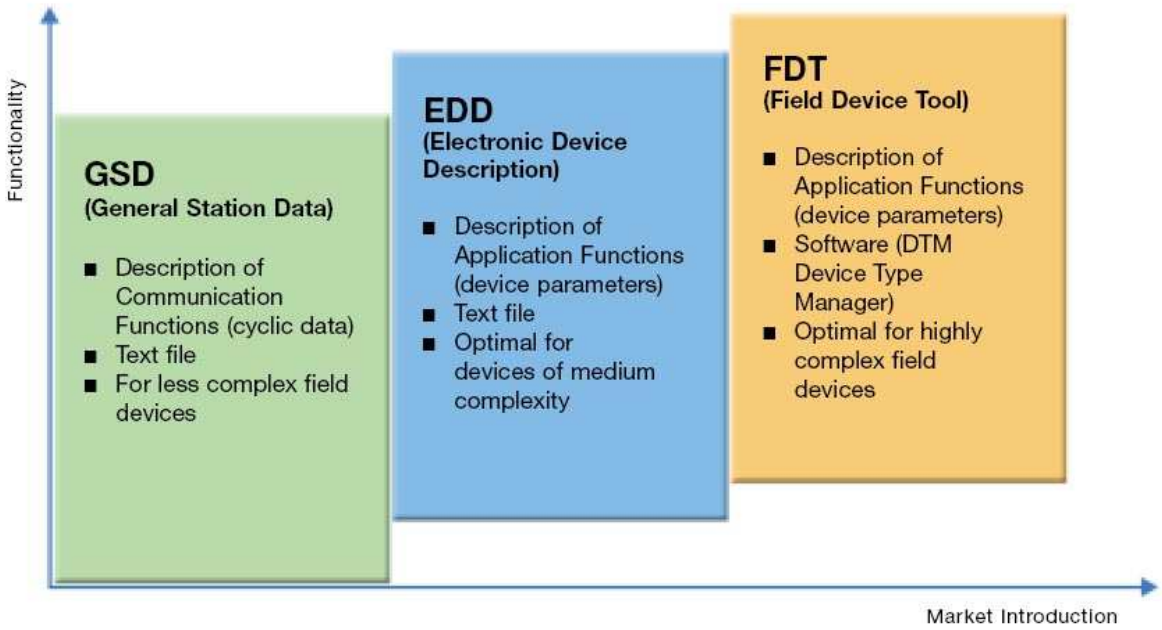
Diplomová práce je rozdělena do třinácti kapitol. V kapitole druhé jsou popsány základní možnosti, jakými lze popisovat provozní přístroje. V kapitole třetí až páté jsou stručně rozebrány základní principy standardu FDT a je v nich popsáno, jaké možnosti tento standard nabízí. Kapitola šestá shrnuje základní znalosti o průmyslové sběrnici PROFIBUS. V kapitole sedm je popsán vývojový nástroj dtmManager společnosti M&M Software GmbH, který byl použit pro tvorbu ovladačů. Samotnou tvorbou obou modulů DTM se zabývají kapitoly osm a devět. V těchto kapitolách je podrobně popsána stavba jednotlivých modulů a jsou zde rozebrány jejich funkce. Kapitola deset popisuje použití modulů DTM v rámcové aplikaci FDT, práci s nimi a jejich přenositelnost na jiný systémy. V příloze je přiložen popis jednotlivých parametrů profilu zařízení pro měření teploty, které pracuje jako PROFIBUS PA slave verze 3.0. Tyto parametry byly implementovány v ovladači zařízení.

# 2. Nástroje pro integraci přístrojů do řídícího systému

## 2.1. Úvod

Velký počet různých typů provozních zařízení uvnitř řídícího systému způsobuje, že konfigurace a správa takovéhoto systému je poměrně obtížná a časově náročná. S technickým rozvojem akčních členů, přístrojů a moderních průmyslových sběrnic vznikla zároveň i snaha o začlenění inteligentních provozních přístrojů do inženýrských a řídících systémů způsobem, který by umožňoval uživatelům snáze spravovat, monitorovat a předávat data do softwarových nástrojů pro konfiguraci a správu.

Základním požadavkem při integraci zařízení do systému je otevřenost a flexibilita – tedy možnost umístit do systému přístroje od různých výrobců. V průměrném řídícím systému je běžně umístěno sto různých provozních přístrojů od deseti různých výrobců, což značí deset různých programů pro správu těchto přístrojů. Cílem integrace zařízení je nahradit všechny tyto programy pouze jedním jediným programem, který by byl schopen spravovat všechna zařízení nezávisle na jejich výrobci. Tato snaha vedla výrobce těchto přístrojů k vytvoření tří navzájem na sebe navazujících konceptů popisu zařízení.




Obrázek 1 - Vývoj nástrojů pro popis zařízení

Prvním nástrojem umožňujícím popis zařízení jsou soubory GSD (General Station Data). Nad technologií GSD stojí technologie EDD (Electronic Device Description) založená na textově orientovaném jazyce EDDL (Electronic Device Description Language). Nejkomplexnější z těchto konceptů je pak standard FDT (Field Device Tool), který pro popis jednotlivých přístrojů začleněných v systémů používá programové komponenty DTM (Device Type Manager).

## 2.2. Standard General Station Data (GSD)

Nejstarším ze standardů pro popis zařízení je GSD soubor. GSD soubor je univerzální popisný nástroj pro konfiguraci zařízení bez ohledu na jeho typ nebo výrobce. Jedná se o textový, člověkem bez problémů čitelný soubor s pevně určenou strukturou povinných a nepovinných parametrů s jasně definovaným datovým typem a limitními hodnotami. Tyto parametry obsahují základní informace o zařízení, které jsou nezbytné k jeho připojení ke komunikační sběrnici a následnému provozu, jako jsou například přenosová rychlost, doba časových odezev, počet I/O signálů, diagnostické hodnoty apod.

GSD soubor je obvykle dodáván výrobcem zařízení, který ručí za jeho funkčnost a správnost. S nově se objevujícími přístroji a rozšiřováním technických možností dochází i k rozšiřování konfiguračních parametrů, základní sada parametrů však zůstává stále stejná.

## 2.3. Standard Electronic Device Description (EDD)

V tomto standardu založeném na popisu přístroje (device description – DD) je využito souboru popisných dat zařízení – EDD (Electronic Device Description). Narozdíl od předcházejícího standardu GSD však nemá EDD soubor striktně definovanou a předepsanou strukturu.

Cílem EDD je umožnit jedinému komunikačnímu nástroji se základní znalostí prostředí získat podrobné informace o samotném zařízení a jeho chování. Výhodou takovéto koncepce je nahrazení celé sady programů, z nichž každý je schopen komunikovat pouze se zařízením, pro který byl navržen.

Data obsažená v souborech EDD jsou čtena a vyhodnocována prohlížecím programem, který je svojí funkcí podobný například internetovému prohlížeči.

Výrobce daného zařízení vytváří soubor EDD pomocí textově orientovaného jazyka EDDL (Electronic Device Description Language), který je stále ještě čitelný člověkem s možností snadné úpravy zdrojového kódu. Tvůrce konfiguračního programu pak již zbývá jen implementovat způsob čtení EDD souboru. Aby tedy bylo možné s EDD soubory pracovat, musí prohlížecí program obsahovat interpreta jazyka EDDL – EDDI (Electronic Device Description Interpreter). Výhodou takovéhoto konceptu je, že pro každý operační systém stačí pouze jediný konfigurační nástroj, neboť samotný EDD je vzhledem ke svému formátu (ASCII) na operačním systému nezávislý.

Jazyk EDDL definovaný mezinárodní normou IEC 61804-2 je v současnosti stále nejpoužívanějším jazykem pro popis provozních přístrojů. Umožňuje přesné vyjádření stavu zařízení, specifikaci dat pro jeho konfiguraci a parametrizaci, pokyny pro manipulaci s těmito daty, diagnostické informace apod. EDDL je schopen popsat formát a význam datových struktur daného zařízení, jeho chování a uživatelské rozhraní, které je nutné pro práci se zařízením. Svojí syntaxí vychází EDDL ze specifikace ANSI C.

Pokud dojde ke změně nebo modifikaci provozního přístroje umístěném v řídícím systému, lze jednoduše vybavit prohlížecí nástroj novým souborem EDD, který překryje zastaralý soubor. Odpadá tak neustálá instalace nových konfiguračních programů, stačí jen nahrát nové EDD soubory.

## 2.4. Standard Field Device Tool/Device Type Manager (FDT/DTM)

Na rozdíl od předchozích standardů GSD a EDD využívajících k popisu zařízení popisných souborů, Field Device Tool (FDT) používá k popisu zařízení programové komponenty známé pod názvem Device Type Manager (DTM).

Podrobněji je tento standard popsán v kapitole 3.

# 3. Specifikace standardu FDT/DTM

## 3.1. Vznik a cíle standardu FDT/DTM

Řešení popisu zařízení založené na programových komponentách zveřejnila poprvé na konci devadesátých let dvacátého století skupina výrobců řídících systémů a akčních členů (ABB, Siemens, Endress + Hauser, Invensys, Metso) se snahou definovat nový standard pro konfigurační nástroj inteligentních přístrojů, který by byl nezávislý na výrobci a typu použitého komunikačního protokolu. V současné době se organizace zabývající se rozvojem standardu FDT nazývá FDT Joint Interest Group (FDT JIG) a jejími členy je několik desítek společností zabývajících se průmyslovou automatizací a řídícími systémy.

Hlavním cílem standardu FDT je dosáhnout následujících bodů:

- Centrální stanice pro diagnostiku, plánování a obsluhu s přímým přístupem ke všem zařízením
- Integrovaná, konzistentní konfigurace a dokumentace řídícího systému, sběrnic a přístrojů
- Jednoduchá organizace dat pro řídící systém a zařízení
- Centrální databáze dat, zálohování a datová bezpečnost
- Snadné a rychlé začlenění různých typů zařízení do řídícího systému

Výrobce zařízení definuje obvykle konfiguraci, služby a diagnostické funkce pro každé své zařízení sám a také vytváří uživatelské rozhraní potřebné k obsluze zařízení v řídícím systému. FDT si klade za cíl nabídnout výrobcům jediné standardizované uživatelské prostředí, které by nabízelo všechny potřebné funkce v integrovaném rozhraní a umožnilo jim tak snížit výrobní náklady spojené s vývojem vlastního rozhraní.

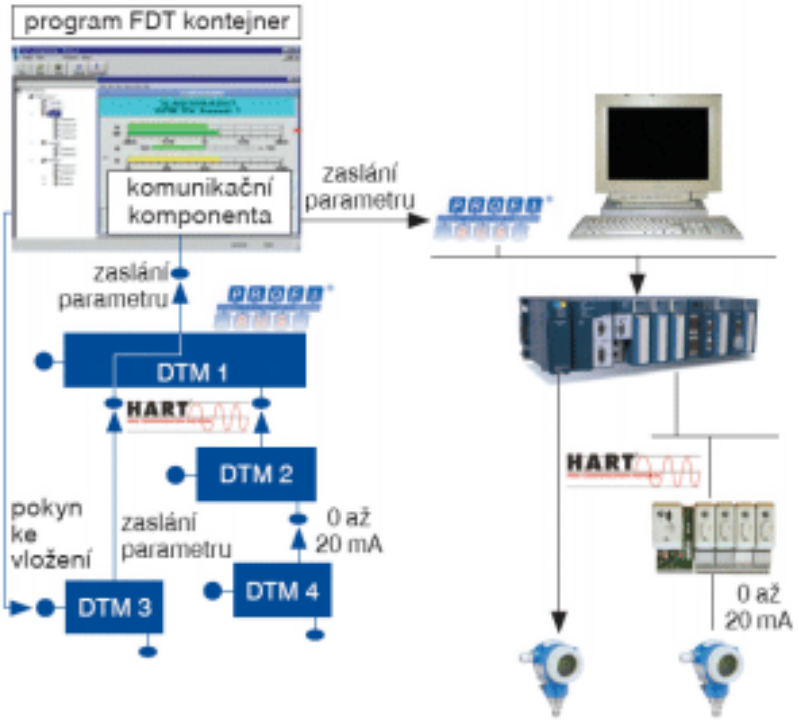
## 3.2. Technologické pojetí

Provázání softwarových modulů s hardwarovými komponentami nově přináší do sběrnicových technologií možnosti zařízení „Plug & Play“. Modul DTM, v němž jsou zapracovány všechny příslušné funkce a specifické údaje o daném zařízení, si obecně můžeme představit jako obdobu ovladače pro tiskárnu nebo jiné podobné periferní zařízení.

Použití technologie FDT vyžaduje implementaci společně s operačním systémem Microsoft Windows, neboť standard FDT je založen na využití objektově orientovaných komponent Microsoft COM (Component Object Model), popřípadě Microsoft DCOM (Distributed Component Object Model). Microsoft Windows rovněž nabízí využití technologie ActiveX, jejíž výhodou je možnost definovat rozhraní obsahující nejenom data, ale i funkce.

Toho může být s úspěchem využito například ve spojení s OPC (OLE for Process Control). Cílem OPC i FDT je vzájemné propojení jednotlivých softwarových a hardwarových komponent obsažených v řídícím systému. Při této systémové integraci umožňující sjednotit izolované části systému v jeden celek je použita architektura klient – server.

Komunikace a výměna dat mezi zařízeními, řídícím systémem a uživatelským prostředím je zprostředkována pomocí dokumentů XML (eXtended Markup Language), které jsou přenášeny přes specifikovaná rozhraní. Díky tomu není problém transparentně pracovat s různými sběrnicemi (v současné době existují XML schémata pro protokoly HART, PROFIBUS a Foundation Fieldbus). V případě použití nového sběrnicového protokolu stačí jen definovat nové schéma XML.




Obrázek 2 - Paralela mezi strukturou softwaru a hardwaru v FDT/DTM

## 3.3. Technologické řešení

Koncepce FDT definuje rozhraní mezi softwarovou komponentou, která představuje a specifikuje dané zařízení, a uživatelským nástrojem spravujícím řídící systém. Softwarová komponenta specifikující přístroj se nazývá Device Type Manager (DTM).

Modul DTM není samostatně spustitelný soubor, ale musí vždy umístěn v rámcové (frame) aplikaci programu FDT. Pro přístup k funkcím DTM skrze rámcovou aplikaci se využívá rozhraní ActiveX. Přes toto rozhraní mohou DTM a rámcová aplikace zpracovávat a přenášet potřebná data a komunikovat.

Při tvorbě DTM není nutné znát cílovou rámcovou aplikaci, protože DTM je na rámcové aplikaci nezávislé. Komunikace DTM je implementována tak, že se uživateli jeví, jako kdyby existovalo přímé datové propojení s přístrojem.

V praxi se rozlišují dva typy modulů DTM. Modul CommDTM je komunikační driver, v němž jsou definována specifická schémata sběrnicového protokolu, a který zajišťuje datové propojení rámcové aplikace s fyzickým systémem. Modul DeviceDTM je driver zařízení, jenž obsahuje podobně jako například ovladač u tiskárny veškeré základní informace popisující daný přístroj nebo inteligentní člen.

Moduly DTM jsou dodávány výrobcem spolu s daným zařízením. Následující vlastnosti jsou pro DTM charakteristické:

- Vícejazyčné uživatelské rozhraní
- Kontrola správnosti parametrů
- Automatické generování závislých parametrů
- Uživatelský návod na komplexní konfigurace, uvádění do provozu, kalibrace apod.
- Čtení a zápis parametrů z/do přístroje
- Diagnostické funkce přizpůsobené danému zařízení
- Generování specifické dokumentace pro daný přístroj
- Žádné přímé propojení s dalším zařízením
- Nezávislost na prostředí
- Podpora pro jeden nebo více typů zařízení

V řídícím systému je ke každému přístroji obsaženém v systému přidružen jeho DTM, který poskytuje přístup ke specifickým vlastnostem daného zařízení. Některá

komplexnější DTM dokáží ovládat i celé typové skupiny zařízení (průtokoměry, tlakoměry, teploměry apod.), které jsou nabízeny výrobcem.

Jak již bylo řečeno, aby bylo možné přistupovat k funkcím DTM, musí být tento modul instalován v rámcové aplikaci FDT, kde vytváří svoji instanci. Rámcová aplikace zajišťuje následující služby:

- Není nutná znalost přístrojů, které chceme připojit do systému
- Ukládání a distribuování dat všech instancí přístrojů pro DTM
- Vytváření komunikačního kanálu a spojení s přístrojem
- Garance konzistence dat v rámci systému
- Řízení činnosti systému v rámci topologie klient – server

# 4. Základy FDT

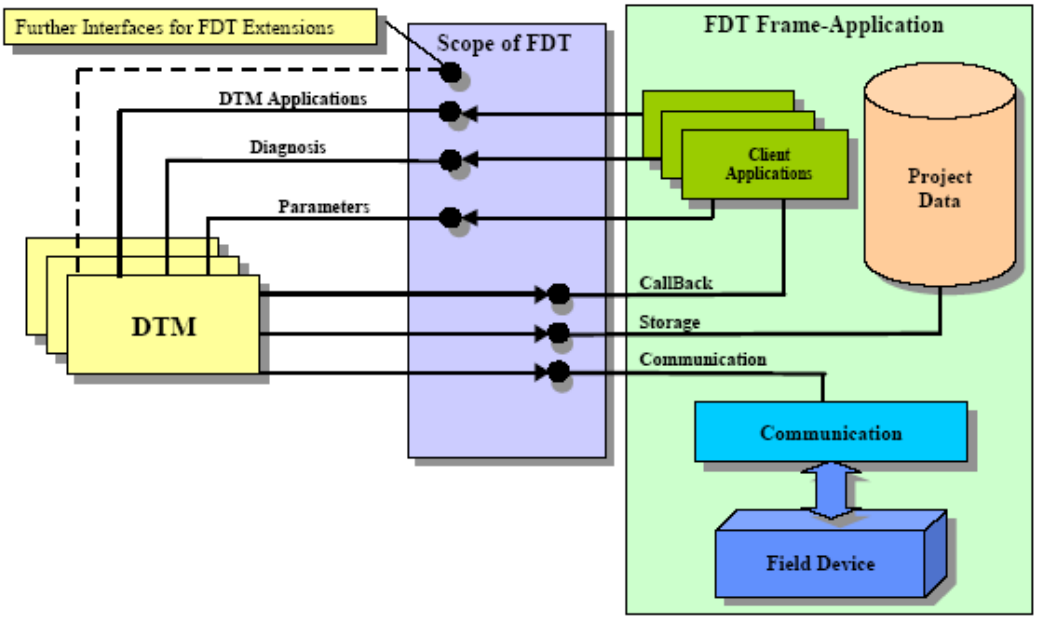
## 4.1. Základní myšlenka FDT

Základní myšlenkou FDT je možnost integrovat do řídícího systému přístroje a zařízení od různých výrobců – každý přístroj má svoji instanci modulu DTM, který se připojuje do monolitické rámcové aplikace, jež spravuje jeho specifické funkce.

V základním pohledu je DTM složeno z několika objektů: objektu serveru a objektů kanálu. V tomto modelu objekt serveru představuje základní funkční tělo modulu, který symbolizuje dané zařízení. Pro použití zařízení musí docházet mezi zařízením a rámcovou aplikací ke komunikaci skrze FDT komunikační kanály. Tyto kanály mohou být procesně identické s I/O vstupy a výstupy zařízení nebo mohou přenášet jiné procesní hodnoty v závislosti na daném zařízení či sběrnici.

## 4.2. Základní architektura FDT

FDT je specifikace rozhraní založená na interakci mezi zařízením a FDT rámcovou aplikací (rovněž se používá výraz FDT kontejner). Typicky se rámcová aplikace skládá z klientských aplikací využívajících funkcí DTM, databáze trvalých dat vztahujících se k zařízením/DTM a komunikační linky k zařízením.




Obrázek 3 - Architektura FDT

Klientské aplikace jsou jednoduché aplikace, které zajišťují speciální služby rámcové aplikace, jako jsou například konfigurace, sledování, správa kanálů apod. Ke své činnosti využívají tyto miniaplikace prostřednictvím rozhraní ActiveX funkce obsažené v modulech DTM.

Do databáze perzistentních dat jsou ukládány informace vztahující se k DTM či zařízení zařazených do projektu. Typicky se jedná především o údaje jako je název přístroje, jeho verze, název výrobce, datum poslední úpravy apod.

Komunikační linka rámcové aplikace FDT umožňuje využívat služby operačního systému takovým způsobem, že se uživateli jeví, jako by měl přímé spojení s daným zařízením a mohl s ním komunikovat.

## 4.3. Základní objekty a rozhraní FDT

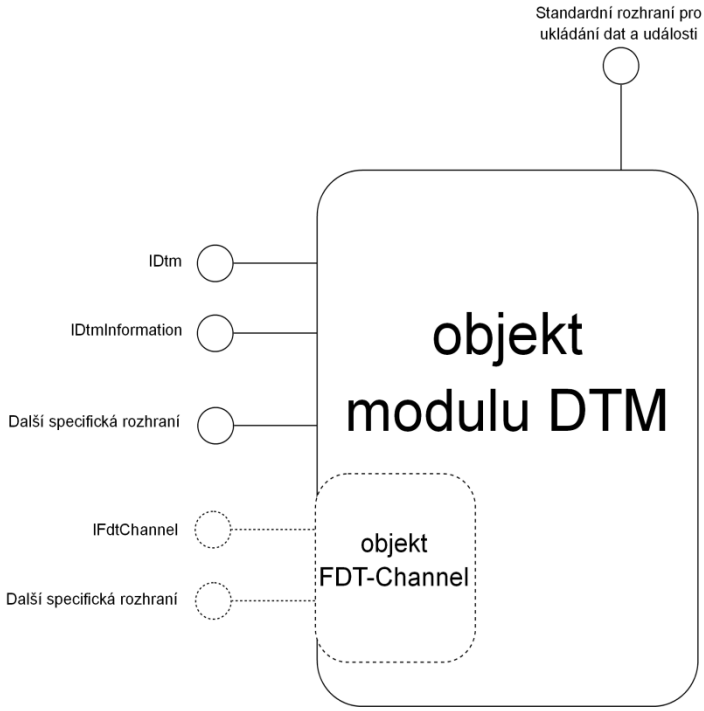
### 4.3.1. Device Type Manager (DTM)

V řídícím systému je běžně umístěno množství různých provozních přístrojů. K obsluze těchto přístrojů v rámci standardu FDT je třeba jednoho či více modulů Device Type Manager (DTM). Moduly DTM jsou instalovány do systému, takže řídící systém může být dynamicky rozšiřován instalací nových DTM pro provozní přístroje, které jsou nově přidány do systému. Stejně tak je možné jednoduše odstranit DTM, pokud je přístroj odebrán.

DTM je možné vytvořit způsobem, že bude spravovat pouze jeden konkrétní typ zařízení, nebo může být jeho struktura navržena tak, že bude schopen spravovat celou skupinu rozdílných zařízení. Obvykle je používána spíše první varianta, kdy jedno DTM spravuje pouze jedno zařízení a zná všechny jeho specifické parametry.

V každém DTM nezávisle na tom, pro co je ve výsledku určen, musí být implementována rozhraní *IDtm* a *IDtmInformation*. Tyto dvě rozhraní obsahují základní funkce a informace pro správnou funkci DTM. Další rozhraní rozšiřují možnosti, jež bude DTM následně poskytovat, a je pouze na rozhodnutí tvůrce modulu, která rozhraní se rozhodne do svého DTM začlenit.

Pro komunikaci a výměnu dat s rámcovou aplikací se do DTM implementuje objekt FDT komunikačního kanálu (FDT-Channel object). Přístup k tomuto kanálu je umožněn skrze rozhraní *IFdtChannel*, které umožňuje přístup ke všem parametrům, jež tento kanál popisují.




Obrázek 4 - Rozhraní modulu DTM

### 4.3.2. Block Type Manager (BTM)

Block Type Manager neboli BTM je softwarový objekt, který může být použit pro reprezentaci vnitřní struktury uvnitř zařízení. BTM vychází ze stejné struktury jako DTM – je spravováno stejným stavovým modelem a jsou v něm implementována podobná rozhraní.

Pro připojení BTM do struktury DTM se používají standardní rozhraní FDT. DTM v takovém případě představuje přístupovou bránu k BTM a pro komunikaci mezi těmito moduly je použit mechanismus vnořené komunikace.

Koncept BTM je nezávislý na použitém sběrnicovém protokolu a může být použit na různé protokoly.

### 4.3.3. Rámcová aplikace FDT

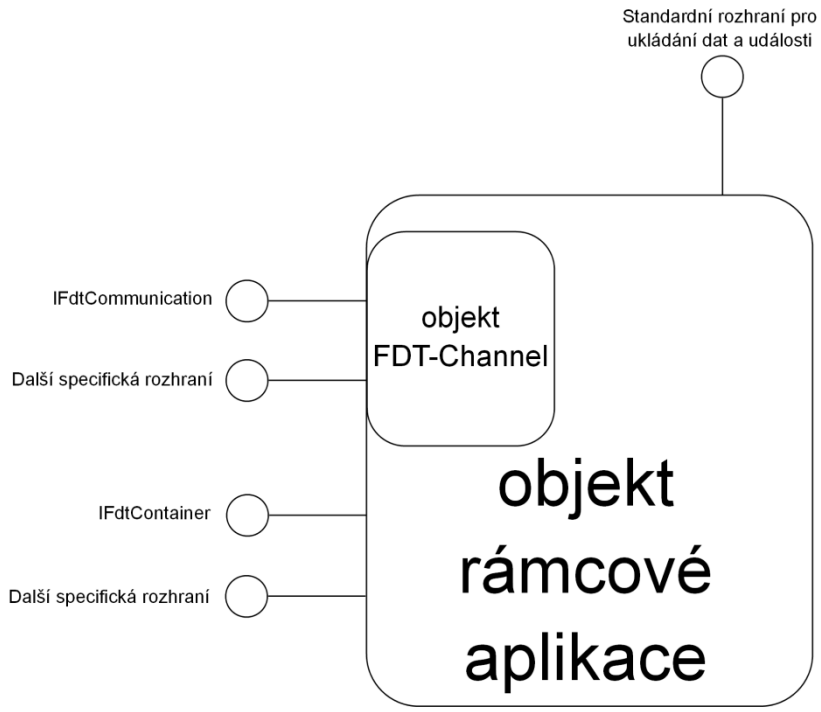
Rámcová aplikace FDT poskytuje prostředí pro správu dat, komunikaci s provozním zařízením a obsluhu modulů DTM a všechny specifické úkoly jsou touto aplikací vyřizovány. Přitom musí být celá tato aplikace postavena tak, aby na ní byla všechna DTM nezávislá a mohla tak být použita v libovolné existující rámcové aplikaci.

Kromě své nejdůležitější funkce, což je přístup ke specifickým funkcím DTM a správa struktury řídícího systému, se rámcová aplikace stará i o správu uživatelských dat, ať už se jedná o jednoduchý systém souborů nebo komplexní databázový systém. Data DTM ukládaná tímto způsobem skrze rozhraní rámcové aplikace jsou pak přímo vázána na DTM a nejsou dostupná pro další aplikace. Tím je zajištěno, že každé DTM bude mít přístup jenom ke svým datům a nemůže tak dojít k situaci, že by jedno DTM mohlo přistupovat k datům jiného DTM.

Přístup DTM k jednotlivým rozhraním, jež nabízí rámcová aplikace, probíhá přes hlavní rozhraní *IFdtContainer*, které zastřešuje vzájemnou interakci mezi rámcovou aplikací a DTM. Doplňující rozhraní, která může DTM využívat, jsou daná v závislosti na konkrétní rámcové aplikací.

DTM v technologii FDT nemá znalosti o topologii řídícího systému a o zařízeních, které jsou spolu s ním do systému připojeny. Z hlediska uživatele se tak jeví, že rámcová aplikace umožňuje přímé propojení (peer-to-peer) mezi DTM a zařízením. Unikátnost každého spojení je nezbytná především při komunikaci v asynchronním režimu a je zajištěna unikátním identifikátorem, který je jedinečný pro každé spojení.

Tato komunikace je podobně jako v případě DTM zajišťována objektem FDT komunikačního kanálu, který komunikuje pomocí rozhraní *IFdtCommunication* a vytváří spojení se zařízením. Bez služeb rozhraní *IFdtCommunication* poskytovaných rámcovou aplikací nedokáže DTM přistupovat na sběrnici a komunikovat se zařízením. Rámcová aplikace si pro účely této komunikace udržuje přehled o topologii řídícího systému pomocí databáze, ke které přistupuje skrze rozhraní *IFdtTopology*.




Obrázek 5 - Rozhraní FDT

## 4.4. Komunikace pomocí XML

Výměna dat mezi FDT rámcovou aplikací probíhá výhradně s použitím XML. Pro každý specifický úkol, jako je například přiřazování komunikačního kanálu, sledování připojených klientů, konfigurace zařízení apod., existuje specifické schéma XML, přes které tato komunikace funguje.

Pro reprezentaci XML dat v aplikaci existují standardizované nástroje, jako je například W3C DOM (W3C's Document Object Model). Použití tohoto nástroje zaručuje validitu syntaxe dokumentu XML a umožňuje tak programátorům vytvořit aplikaci, která bude platformě nezávislá a bude fungovat na všech prohlížečích a serverech. Objekt DOM lze v prostředí operačního systému Microsoft Windows vytvořit například pomocí XML Parseru. Výhody použití DOM jsou následující:

- XML Parser generuje DOM z přenesených XML dat
- Schémata zaručují validní DOM se správně definovanými elementy
- DOM nabízí standardní metody pro přístup k datům umístěným ve stromové struktuře
- DOM dokáže generovat XML data pro přenos dat

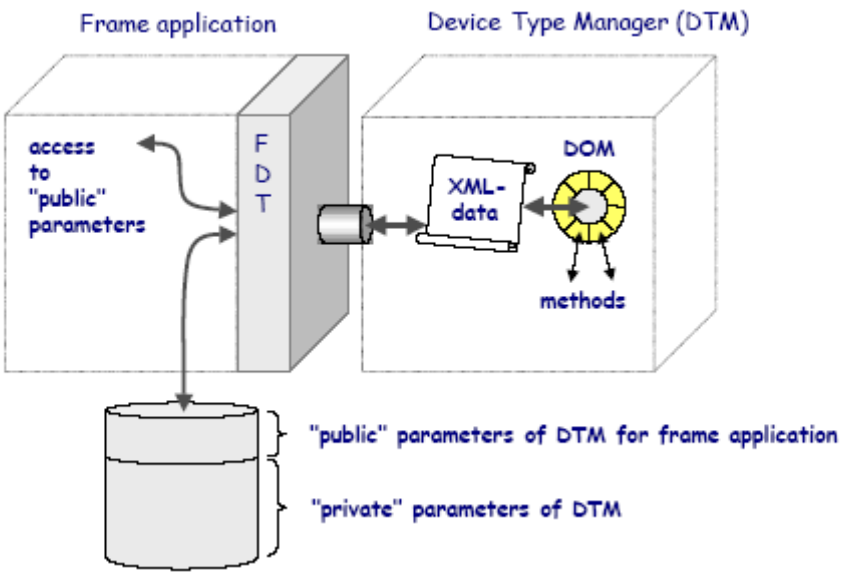
Velkou výhodou je v takovémto případě synchronizace přístupu k datům – obecně je v takovýchto případech problém v tom, že chceme-li číst nebo zapisovat nějaká data, musí bát tato operace provedena jako jediná transakce, aby si běžící aplikace navzájem nepřepisovala data. Při komunikaci pomocí schémat XML jsou všechny elementy obsažené v XML dokumentu přenášené v jediné transakci, tudíž je synchronizace zaručena.

Dokument XML používaný v FDT pro komunikaci obsahuje data zařízení a informace nezbytné pro navázání spojení mezi DTM a zařízením, jako jsou například informace o typu sběrnice, adresa, na které je zařízení v řídícím systému umístěno apod.

## 4.5. Uchovávání trvalých dat v FDT

Data zpracovávaná v FDT mohou být dvou různých typů: První druh dat pochází přímo od modulů DTM. Druhý typ dat se vztahuje k datům rámcové aplikace spravující projekt, v kterém je umístěn řídící systém.

DTM, které nemá implementováno vlastní ukládání dat, musí být schopné ukládat data a pracovat s nimi v databázi vyhrazené danému projektu v rámcové aplikaci. V takovém případě se běžně používá COM rozhraní *IPersistXXX* a *IPersistPropertyBag* nebo *IPersistStreamInit*. Správu dat přebírá v tomto případě rámcová aplikace, které synchronizuje přístup k datům pomocí rozhraní *IFdtContainer*.




Obrázek 6 - Uchovávání a přístup k datům v FDT

DTM, které pro správu dat používají vlastní systém ukládání, musí pro přístup k těmto datům použít vlastního rozhraní. Nevýhodou toho je, že programátor musí toto rozhraní sám vytvořit a implementovat. Naopak výhodou tohoto systému je, že je zaručena konzistence dat, neboť do databáze přidružené k DTM nemají jiné aplikace přístup. Použije-li programátor tuto variantu, musí vytvořit rozhraní pro správu dat, které bude předávat do rámcové aplikace, takovým způsobem, aby všechny nezbytné parametry odpovídaly rozhraní *IPersistXXX*. Pro interní data lez použít rozhraní *IDtmExportImport*.

## 4.6. Přístup na sběrnici

Nezávislost integrace DTM do rámcové aplikace je realizována pomocí specifické identifikace sběrnic. Tato identifikace se označuje jako UUID. Identifikátory UUID jsou používány jako parametry metod rozhraní a podle hodnoty UUID, jež je zaznamenána v XML dokumentu, dochází ke specifické interakci mezi DTM a rámcovou aplikací.

Specifické informace o sběrnici jsou spolu s jejím UUID uloženy v DTM v XML dokumentu, jako je například DTMParameterSchema nebo FDTHARTChannel-ParameterSchema a jsou dostupné přes rozhraní příslušné k dané sběrnici. Informace o podporovaných sběrnicích mohou být využity například při ověřovaní topologie sběrnice nebo při připojování během komunikace.

V takto fungujícím mechanismu stačí rámcové aplikaci znát pouze UUID přímo připojených sběrnic. Pro sběrnice, které nejsou přímo připojeny, stačí, aby o nich byl záznam v příslušném XML dokumentu.

## 4.7. Operační fáze a přístupová práva

FDT umožňuje nastavit rozdílná přístupová práva pro různé typy uživatelů. Tato přístupová práva vymezují, ke kterým funkcím lze přistupovat, a určují tak roli uživatele. Použitím těchto práv lze přesně definovat volnost a interakci uživatele. To zaručuje zvýšení bezpečnosti při řízení a údržbě rozsáhlého řídícího systému.

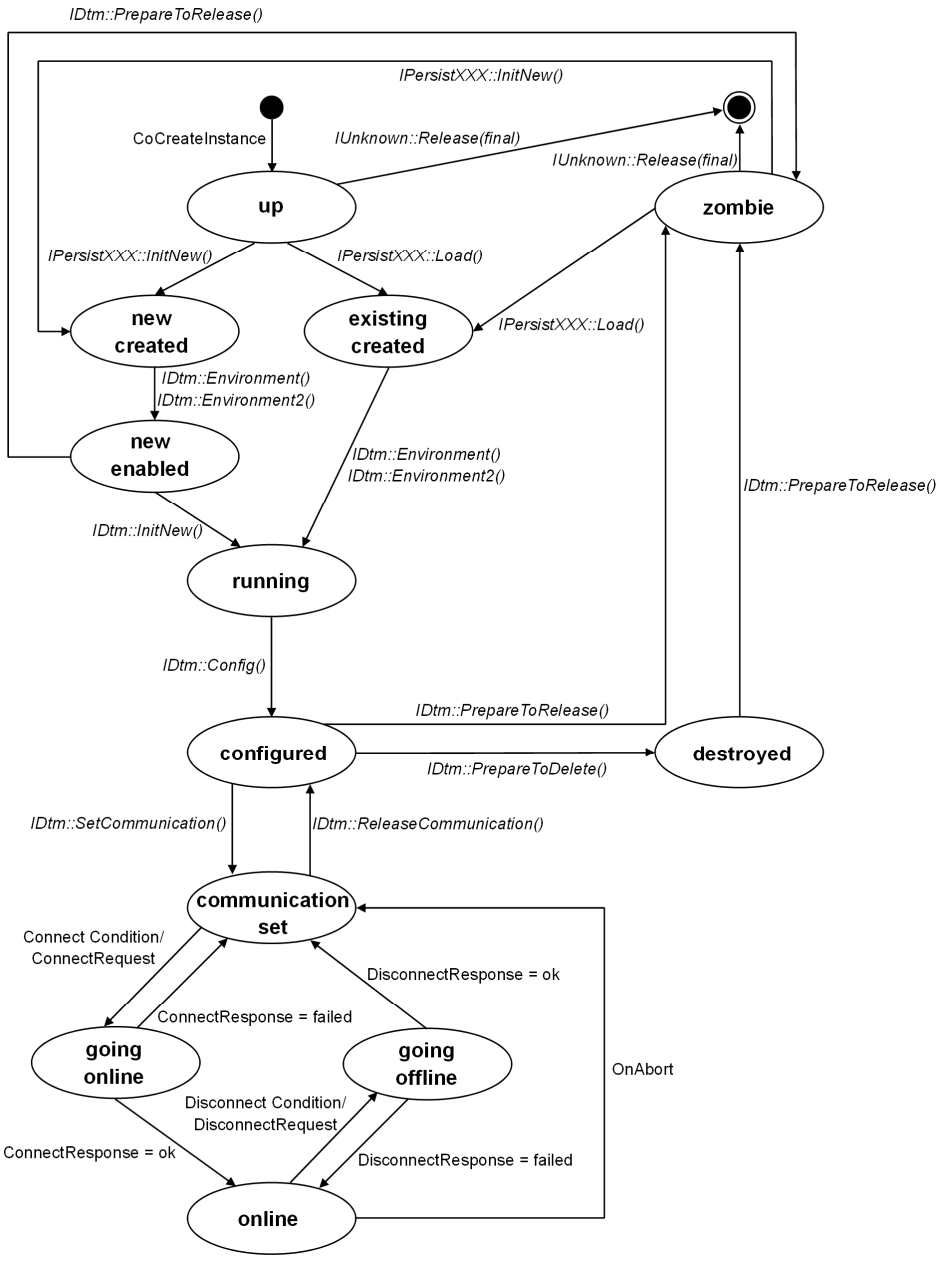
Přístupová práva určují, na jakých operačních fázích se může uživatel podílet. Existují čtyři rozdílné typy operačních fází:

- **Operační fáze Engineering**
  Konfigurace a rozvrhování řídícího systému.
- **Operační fáze Commissioning**
  Instalace zařízení do systému, skenování sítě, diagnostika atd.
- **Operační fáze Runtime**
  Sledování systému za běhu, prohledávání sítě za účelem ověření struktury, čtení procesních hodnot a diagnostických informací, velice omezená možnost konfigurace zařízení.
- **Operační fáze Service**
  Neomezený přístup k zařízení, všechny operace související s údržbou.

# 5. Základní koncepty FDT

## 5.1. Stavové schéma DTM

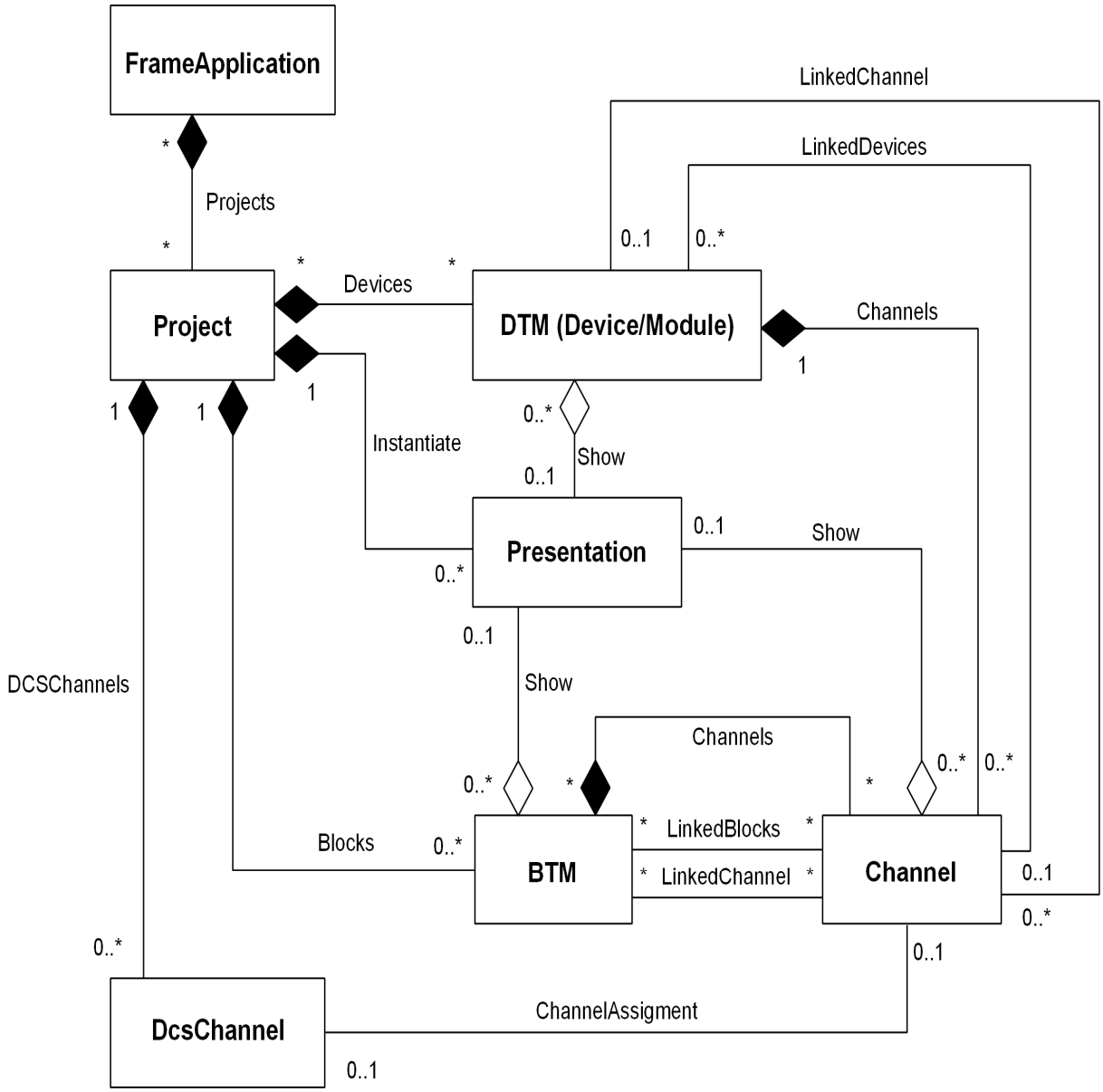
DTM může v systému nabývat různých stavů. Všechny tyto stavy a možnosti, jak jich DTM může dosáhnout, jsou zachyceny ve stavovém schématu DTM. Horní část schématu zachycuje stavy DTM při inicializaci a aktivaci (popřípadě deaktivaci), ve spodní části schématu jsou pak zakresleny jednotlivé stavy při komunikaci.




Obrázek 7 - Stavové schéma DTM

## 5.2. Objektový model FDT

Tento model zachycuje základní principielní strukturu FDT. Objektové modely DTM, BTM a komunikační kanál FDT slouží k přístupu k informacím. Objekt rámcové aplikace slouží ke konfiguraci a správě komunikačních kanálů. DTM, BTM a komunikační kanály jsou implementovány jako COM objekty a uživatel k nim může přistupovat skrze specifická rozhraní přes objekt serveru. Všechny ostatní objekty a asociační vazby jsou vytvořeny jako dokumenty XML.




Obrázek 8 - Vztahy mezi objekty v FDT

### 5.2.1. Objekt rámcové aplikace (FrameApplication)

Objekt rámcové aplikace je logický objekt reprezentující prostředí, v kterém probíhá správa řídícího systému v podobě obsluhy instancí DTM uvnitř projekt. Rámcová aplikace může naráz obsluhovat více různých projektů.

### 5.2.2. Objekt projektu (Project)

Objekt projektu je logický objekt popisující strukturu, řízení a správu instancí zařízení uvnitř rámcové aplikace. V praxi to znamená především obsluhu instancí dat příslušejících jednotlivým zařízením v databázi nebo systému souborů používaných pro ukládání dat. Projekt je součástí rámcové aplikace.

### 5.2.3. Objekt DTM

V FDT je každé zařízení reprezentováno pomocí objektu DTM, který slouží k popisu zařízení. Každé zařízení může být rozděleno na různé moduly, které mohou být hardwarové nebo softwarové. Hardwarové moduly jsou umístěn ve fyzických slotech zařízení. Softwarové moduly představují statické nebo konfigurovatelné substruktury zařízení. Tyto softwarové moduly jsou pro každé zařízení specifické a nemohou být beze ztráty funkčnosti ze zařízení odebrány.

DTM mohou představovat různé druhy zařízení, jako jsou například provozní přístroje (field devices) a komunikační zařízení (communication devices):

- **DeviceDTM** reprezentují objekty klasických provozních přístrojů, jako jsou například teploměry, čidla tlaku, hladiny apod. Pro komunikaci se svými fyzickými protějšky používají komunikačních kanálů.
- **CommDTM** reprezentuje objekt komunikačního přístroje, jako je například komunikační karta v PC. CommDTM je přístupový bod, přes který se DeviceDTM připojují do fyzického systému a který zajišťuje komunikaci.

### 5.2.4. Objekt BTM

Objekty BTM se používají pro reprezentaci flexibilních softwarových objektů, jako jsou například funkční bloky zařízení, ale lze s nimi modelovat blokovou strukturu zařízení. Výhodu těchto bloků je větší flexibilita než u klasických

softwarových modulů, neboť mohou být podle potřeby přesouvány mezi zařízeními.

### 5.2.5. Objekt kanálu FdtChannel

Objekt kanálu FdtChannel může mít dvě variace v podobě komunikačního kanálu nebo kanálu procesního:

- **Komunikační kanál** je objekt zajišťující přístup do komunikační infrastruktury a obvykle je používán DTM pro správu komunikační linky.
- **Procesní kanál** reprezentuje procesní hodnoty, jako jsou například zvláštní informace o stavu zařízení (alarmy apod.).

### 5.2.6. Objekt prezentace (Presentation)

Objekt prezentace reprezentuje viditelné uživatelské rozhraní. Může to být například nějaký ActiveX prvek, který uživateli poskytuje možnost pracovat s nabízenými funkcemi DTM.

### 5.2.7. Objekt DcsChannel

DcsChannel je logický objekt představující část DCS funkce. K zpřístupnění vstupů a výstupů, které jsou spravovány cyklicky, je třeba v rámcové aplikaci asociovat I/O funkční bloky s procesními hodnotami zařízení. Vstupy a výstupy I/O funkčních bloků jsou reprezentovány pomocí DCS kanálů, procesní hodnoty pomocí Fdt kanálů.

### 5.2.8. Asociační vazby v modelu

V objektovém modelu jsou kromě jednotlivých objektů zachyceny i asociační vazby mezi těmito objekty. Význam těchto vazeb je popsán níže:

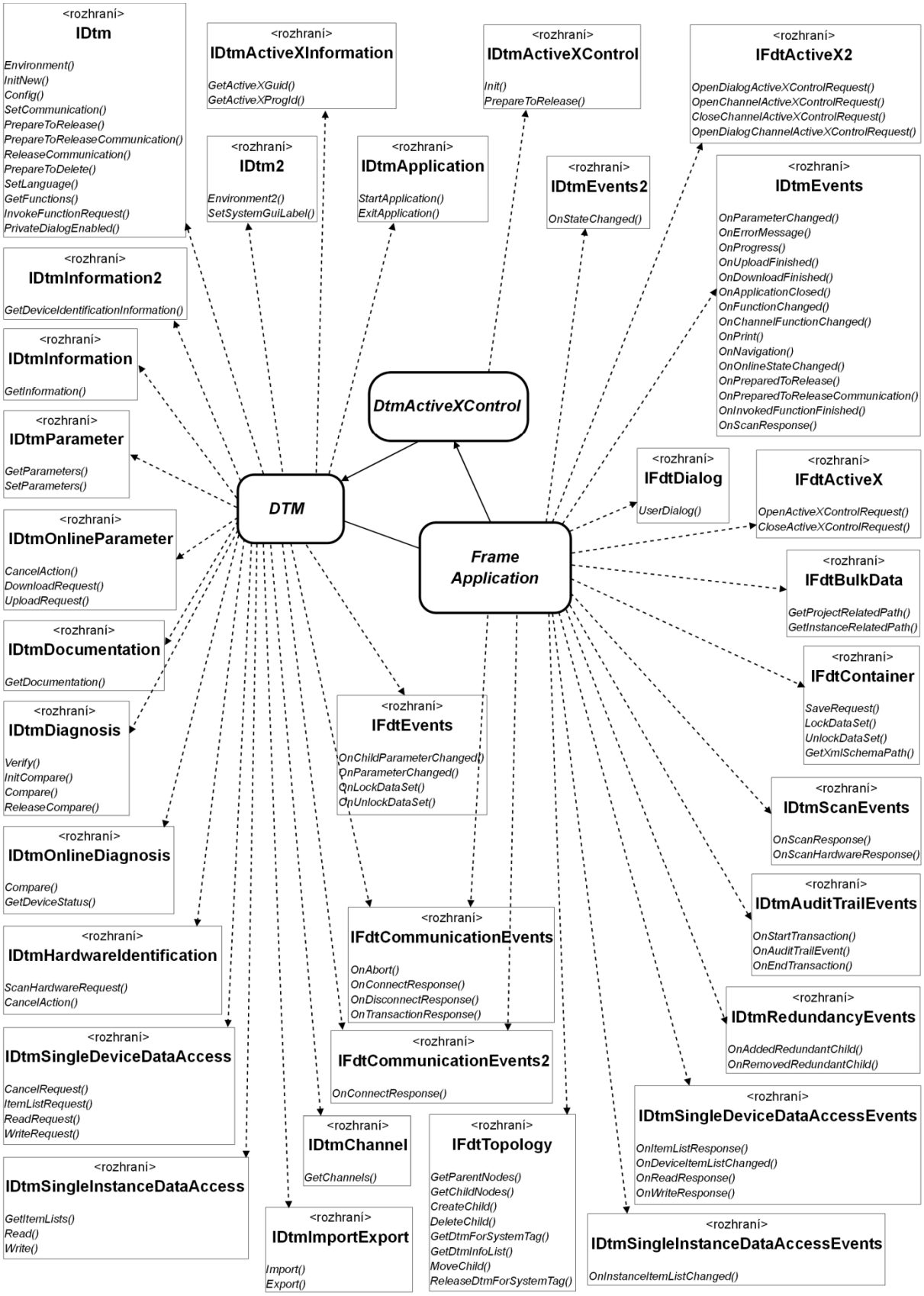
- **Asociační vazba Devices** – Uvnitř projektu je důležitá znalost všech provozních přístrojů. Touto asociací jsou označeny všechny provozní přístroje v projektu a pomocí ní je umožněno modulům DTM s vlastní databáze spolupracovat s projektem při vytváření nebo odstraňování instancí zařízení. Instance DTM, která v projektu představuje skutečný provozní přístroj, nemůže být v jeden okamžik součástí více

než jednoho projektu. V každém projektu musí být vytvořena samostatná instance daného zařízení.

- **Asociační vazba Channels** – Komunikační kanál je nedílnou součástí každého DTM. Instance komunikačního kanálu je vytvářena a spravována modulem DTM.
- **Asociační vazba Show** – Instance prezentace (objekt Presentation) je propojena s jádrem DTM pomocí této vazby, která umožňuje zobrazovat vybrané funkce DTM.
- **Asociační vazba Instantiate** – Ta**to vazební asociace oznamuje ActiveX prvkům, že prezentační objekt** DTM je vkládán a spravován rámcovou.
- **Asociační vazba ChannelAssigment** – K vytvoření cyklických I/O dat dostupných uvnitř rámcové aplikace musí existovat vazba mezi I/O funkčními bloky a procesními hodnotami zařízení. I/O funkční bloky jsou reprezentovány pomocí objektu DcsChannel, procesní hodnoty objektem FdtChannel a asociační vazba mezi těmito objekty je nazývána ChannelAssigment. Za správu této vazby je odpovědná rámcová aplikace.
- **Asociační vazba LinkedDevice** – Topologie zařízení připojených na sběrnici je zachycena pomocí této vazební asociace. Uvnitř topologické struktury je pro zařízení rodičovským uzlem uzel Channel object a vazba LinkedDevice představuje spojení mezi tímto rodičovským uzlem a zařízením- Za správu této vazby je odpovědná rámcová aplikace.
- **Asociační vazba LinkedChannel** – Tato vazba je v mnohém podobná jako předcházející vazba LinkedDevice. Uvnitř topologické struktury je DTM jakožto zástupce provozního zařízení potomkem komunikačního kanálu a tato vazba ukazuje spojení mezi zařízením a objektem komunikačního kanálu.

## 5.3. Rozhraní pro DTM, BTM a rámcovou aplikaci

Rozhraní jsou funkční mechanismy pro začlenění aplikace spravující provozní přístroj do rámcové aplikace. Všechna rozhraní jsou vytvořena jako COM objekty a klient k nim může k danému rozhraní přistupovat voláním na objektu serveru. Tak je klient schopen určit dostupnost rozhraní každého objektu za běhu systému.




Obrázek 9 - Rozhraní pro DTM a rámcovou aplikaci

Každé rozhraní plní v konceptu FDT svoji specifickou roli a nabízí uživateli různé funkce. Některá rozhraní jsou pro chod, komunikaci a začlenění provozního přístroje do systému nezbytná a musí být uživatelem implementována, jiná jsou nepovinná a uživatel je využívat nemusí. Tato nepovinná rozhraní pouze nabízí nadstandardní funkce, které však pro chod aplikace nejsou nezbytné.

Všechna rozhraní FDT jsou konečná a nebudou v budoucnosti měněna. Eventuální další rozšíření jsou založena na dalších nepovinných rozhraních.

### 5.3.1. Rozhraní pro DTM

V následujícím přehledu jsou zachycena všechna důležitá rozhraní pro moduly DTM (bližší informace je možné nalézt například v [1]):

- **Rozhraní IDtm**
  Toto rozhraní je hlavním rozhraním, které je nezbytné pro každé DTM, neboť pomocí něho dochází k inicializaci DTM. Rámcová aplikace využívá tohoto rozhraní k nastavování základních informací, jako je například komunikační rozhraní, pro které je DTM určeno, volba řeči apod., nebo k zisku statických informací jako je verze DTM, výrobce atd.
- **Rozhraní IDtm2**
  Rozhraní IDtm2 rozšiřuje funkce IDtm pro verzi FDT 1.2.1 a vyšší. Toto rozhraní je nepovinné a je možné používat pouze rozhraní IDtm.
- **Rozhraní IDtmActiveXInformation**
  Toto rozhraní umožňuje uživatelskému rozhraní DTM, aby vystupovalo v rámcové aplikaci jako prvky ActiveX.
- **Rozhraní IDtmApplications**
  V tomto rozhraní jsou obsaženy funkce, které umožňují spuštění uživatelského rozhraní. Rozhraní je součástí DTM a nemůže být vloženo do rámcové aplikace.
- **Rozhraní IDtmChannel**
  Rozhraní IDtmChannel slouží k přístupu k objektu komunikačního kanálu. Funkce tohoto rozhraní slouží k přenosu informací, které jsou nezbytné k vytvoření asociačních spojení mezi vstupně/výstupními kanály zařízení a funkcemi v rámcové aplikaci. Funkce rozhraní jsou rovněž použity při vytváření vnořené komunikace v DTM.

- **Rozhraní IDtmDiagnosis**
  Toto rozhraní nabízí základní diagnostické funkce požadované rámcovou aplikací pro DTM s konfiguračními parametry.
- **Rozhraní IDtmOnlineDiagnosis**
  Toto rozhraní nabízí funkce používané rámcovou aplikací pro diagnostiku a validaci sběrnicových systémů za provozu.
- **Rozhraní IDtmInformation**
  Rozhraní IDtmInformation je druhým, které je pro DTM nezbytné a tvoří jeho základ. S použitím tohoto rozhraní se je možné dotazovat na statické informace, jako je verze, výrobce apod. k začlenění DTM do knihoven rámcové aplikace.
- Rozhraní IDtmInformation2
  Toto rozhraní rozšiřuje předchozí rozhraní o nové funkce použitelné ve verzi FDT 1.2.1 a vyšší. Rozhraní je nepovinné.
- **Rozhraní IDtmOnlineParameter**
  Pomocí funkcí tohoto rozhraní může rámcová aplikace za běhu přistupovat k provoznímu zařízení.
- **Rozhraní IDtmParameter**
  Toto rozhraní umožňuje přistupovat rámcové aplikaci k parametrům zařízení. Každé DTM nabízí sadu aktuálních dat vztahujících se k přístroji a záleží pouze na daném přístroji a typu sběrnice, které parametry budou dostupné.
- **Rozhraní IFdtCommunicationEvents**
  Toto rozhraní obsahuje sadu funkcí určených k reakci na vnější události při komunikaci.
- **Rozhraní IFdtCommunicationEvents2**
  Rozhraní obsahující nové rozšiřující funkce pro reakce na události při komunikaci pracující ve verzi FDT 1.2.1 a vyšší.
- **Rozhraní IFdtEvents**
  Rozhraní obsahující sadu funkcí pro reakce na události v rámcové aplikaci.
- **Rozhraní IDtmHardwareIdentification**
  Funkce obsažené v tomto rozhraní jsou použity rámcovou aplikací k určení, zda je dostupný specifický komunikačního hardware. Toto rozhraní je implementováno především v CommDTM k detekci

odpovídajícího komunikačního hardwaru, jako jsou například komunikační karty apod.

- **Rozhraní IDtmSingleDeviceDataAccess**
  Pomocí tohoto rozhraní může rámcová aplikace za běhu asynchronně přistupovat ke specifickým parametrům zařízení. Pro přístup k parametrům jsou použita samostatná vlákna.
- **Rozhraní IDtmSingleInstanceDataAccess**
  Pomocí tohoto rozhraní přistupuje rámcová aplikace ke specifickým datům zařízení v režimu offline.
- **Rozhraní IDtmActiveXControl**
  V rozhraní IDtmActiveXControl je umístěno rozšíření pro standardní prvky ActiveX. S pomocí funkcí tohoto rozhraní je umožněno spojení objektu DTM s ovládacími prvky ActiveX.

### 5.3.2. Rozhraní pro rámcovou aplikaci

V následujícím přehledu jsou zaznamenány rozhraní využívané rámcovou aplikací (bližší popis lze nalézt například v [1]):

- **Rozhraní IDtmEvents**
  Toto rozhraní obsahuje funkce reagující na základní události rámcové aplikace, jako je například zavření aplikace, reakce na požadavek tisku, reakce na chybu, vyvolání funkce apod.
- **Rozhraní IDtmEvents2**
  Rozšíření předcházejícího rozhraní o nové funkce použitelné ve verzi FDT 1.2.1 a vyšší.
- **Rozhraní IDtmScanEvents**
  Rozhraní obsahující funkce reakcí na události, které jsou DTM volány, když vrací skenované informace.
- **Rozhraní IFdtActiveX**
  Toto rozhraní musí být v DTM implementováno, pokud má rámcová aplikace podporovat uživatelské GUI (Graphic User Interface) daného DTM.
- **Rozhraní IFdtActiveX2**
  Toto rozhraní rozšiřuje předcházející rozhraní o nové funkce použitelné ve verzi FDT 1.2.1 a vyšší.

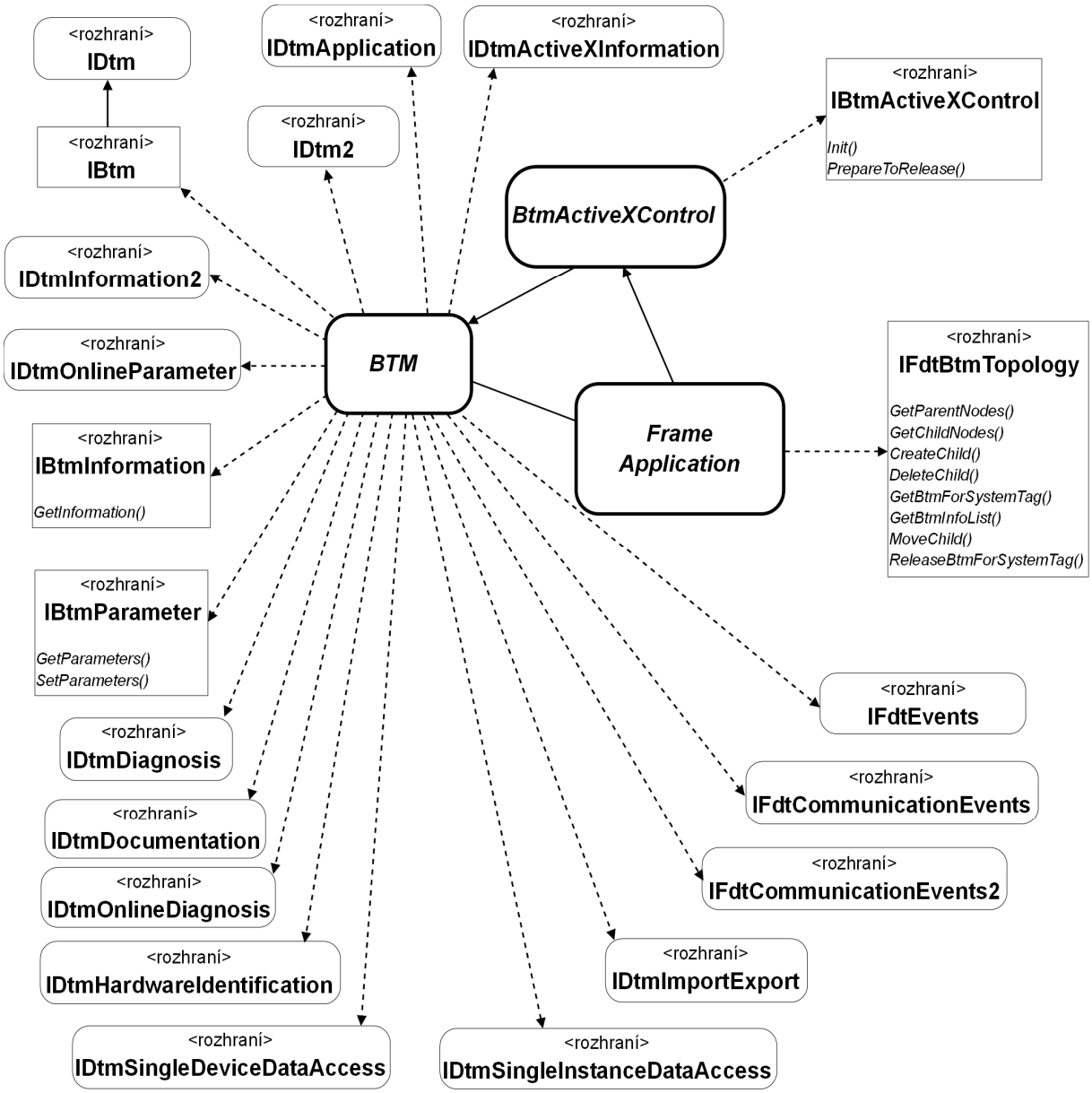
- **Rozhraní IFdtBulkData**
  Rozhraní IFdtBulkData nabízí DTM možnost ukládat velké množství dat jako jsou například sady naměřených hodnot, historii konfiguračních změn apod. do privátních struktur. Je na uživateli, aby zaručil konzistenci těchto dat, rámcová aplikace za to není zodpovědná.
- **Rozhraní IFdtContainer**
  Rozhraní IFdtContainer je hlavním rozhraním pro rámcovou aplikaci a podporuje funkce pro správu dat v multi-uživatelském systému.
- **Rozhraní IFdtDialog**
  Pomocí tohoto rozhraní umožňuje rámcová aplikace zobrazovat DTM zprávy jako chybová oznámení, varování a informace.
- **Rozhraní IFdtTopology**
  Funkce tohoto rozhraní nabízí přístup ke kompletní topologii systému a manipulaci s jejími jednotlivými prvky.
- **Rozhraní IFdtBtmTopology**
  Toto rozhraní má stejné použití jako rozhraní předcházející s tím rozdílem, že skrze něj jde přistupovat k topologii blokových objektů. Obsahuje-li DTM komunikační kanály podporující DTM i BTM, rozhraní IFdtBtmTopology bude poskytovat informace jenom o BTM. Informace vztahující se k DTM lze získat přes rozhraní IFdtTopology.

### 5.3.3. Rozhraní pro BTM

BTM je svými vlastnostmi téměř stejné jako DTM. BTM klasicky využívá stejná rozhraní jako DTM, pouze některá vybraná rozhraní jsou nahrazena rozhraními specifickými přímo pro BTM (bližší popis lze nalézt v [1]):

- **Rozhraní IBtm**
  Rozhraní IBtm je hlavní rozhraní pro BTM. Metody obsažené v tomto rozhraní mají stejnou funkci jako v DTM, jediný rozdíl je, že jsou aplikovány na objekt BTM.
- **Rozhraní IBtmInformation**
  Toto rozhraní nahrazuje pro BTM podobné rozhraní IDtmInformation používané pro DTM. Rozdíl mezi BTM a DTM je zachycen v novém schématu XML.

- **Rozhraní IBtmParameter**
  Pomocí tohoto rozhraní je umožněno rámcové aplikaci přistupovat k parametrům BTM. Podobně jako v předchozích případech funkce rozhraní IBtmParameter korespondují s odpovídajícími funkcemi pro DTM.
- **Rozhraní IBtmActiveXControl**
  Toto rozhraní je rozšířením standardních prvků ActiveX a umožňuje propojení objektu BTM s prvky ActiveX.

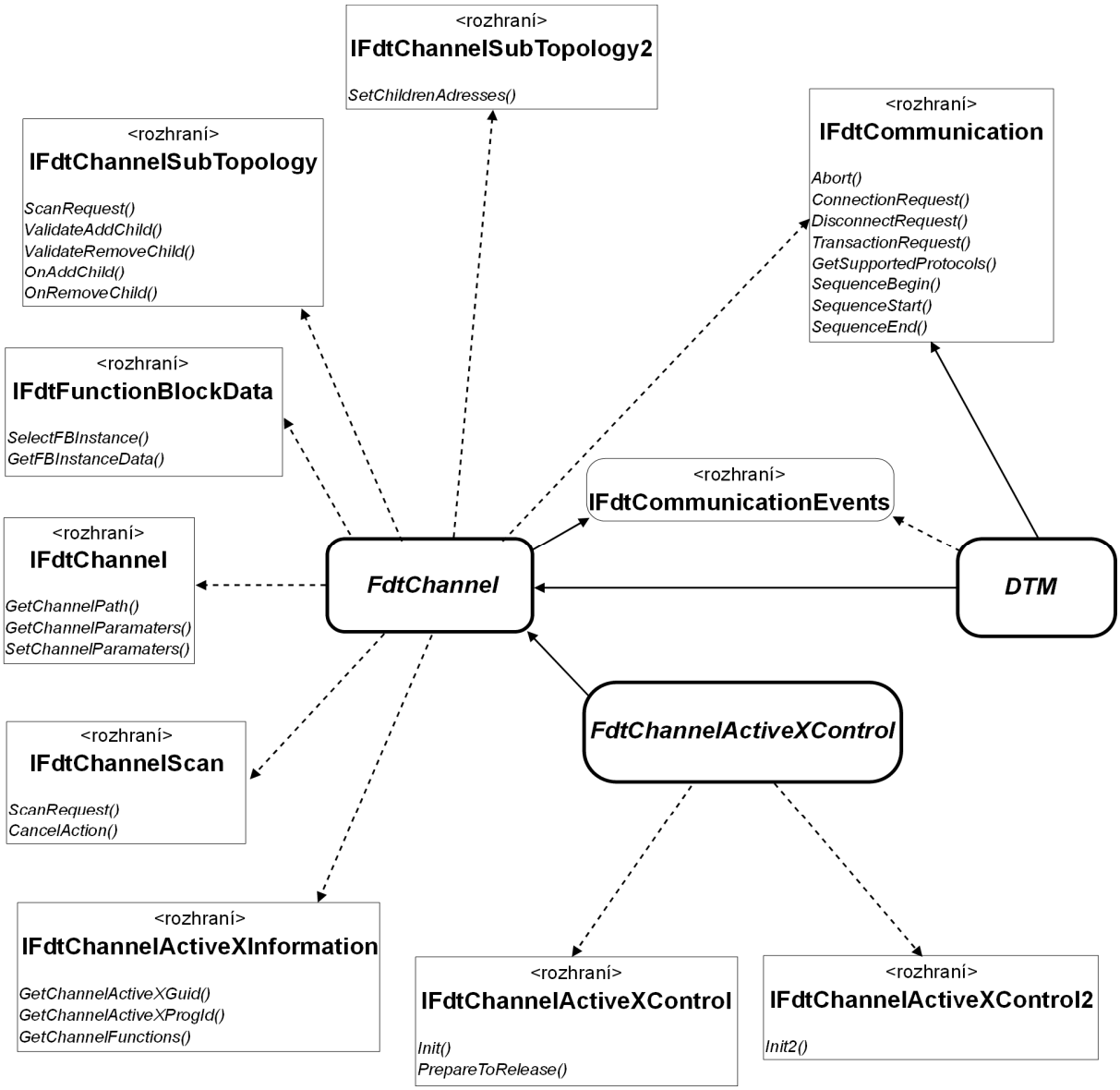



Obrázek 10 - Rozhraní pro BTM

### 5.3.4. Rozhraní pro FDT Channel

Pro komunikaci a výměnu dat mezi rámcovou aplikací, BTM a DTM slouží rozhraní vztahující se k objektu FdtChannel. Tato rozhraní se rovněž zabývají správou a údržbou jednotlivých komunikačních kanálů. V následujícím přehledu jsou popsány jednotlivá rozhraní pro FDT Channel (bližší popis lze nalézt například v [1]):

- **Rozhraní IFdtChannel**
  Rozhraní IFdtChannel je hlavní rozhraní komunikačních kanálů, které slouží ke zjištění informací o kanálu a nastavení jeho parametrů.
- **Rozhraní IFdtChannelActiveXInformation**
  Obvykle objekt FDT komunikačního kanálu nemá žádné uživatelské rozhraní. Přesto však může být nezbytné v závislosti na protokolu sběrnice a možnostech komunikace implementovat uživatelské rozhraní ke konfiguraci komunikace. Běžným příkladem údajů, které musí nastavit uživatel, jsou například nastavení COM portu, ke kterému je zařízení připojeno, nebo nastavení adresy přerušení. Ke správě GUI pro FdtChannel slouží právě toto rozhraní.
- **Rozhraní IFdtCommunication**
  Rozhraní IFdtCommunication vytváří vstupní bod pro kanály s komunikační funkcí. Spojení tohoto rozhraní s komunikační komponentou tvoří základ pro zanořenou komunikaci. Komunikační kanál je schopen podporovat množství rozdílných sběrnicových protokolů. XML dokumenty specifické pro dané protokoly jsou mezi komunikačním kanálem a připojeným klientem vyměňovány pomocí rozhraní IFdtCommunication a IFdtCommunicationEvents.
- **Rozhraní IFdtChannelSubTopology**
  Toto rozhraní slouží ke kontrole vnitřní topologie komunikačního kanálu. Za topologii kanálu je vždy zodpovědná rámcová aplikace, která spolu s příslušným DTM může volat funkce tohoto rozhraní pro kontrolu nastavení spojení.
- **Rozhraní IFdtChannelSubTopology2**
  Rozhraní IFdtChannelSubTopology2 přidává k předchozímu rozhraní možnost nastavovat adresy uvnitř vnitřní topologie a je přístupné od verze FDT 1.2.1. a vyšší.




Obrázek 11 - Rozhraní pro FDT Channel

- **Rozhraní IFdtChannelScan**
  Toto rozhraní obsahuje funkce pro skenování struktury komunikačního kanálu. Tyto funkce nahrazují pro verzi FDT 1.2.1 obdobné funkce obsažené v rozhraní IFdtChannelSubTopology a měly by být pro komunikační kanál podporující danou verzi použity místo nich.
- **Rozhraní IFdtFunctionBlockData**
  Pokud je DTM určeno pro zařízení se zabezpečním proti poruchám, pak musí podporovat toto rozhraní. Je důležité, aby v systému, který podporuje zabezpečení proti poruchám, podporovaly toto zabezpečení i všechny komunikační komponenty a komunikační kanály musí

být schopny rozšířit hlášení o volání funkcí vztahujících se k tomuto zabezpečení.

- **Rozhraní IFdtChannelActiveXControl**
  Toto rozhraní je rozšířením standardních prvků ActiveX a umožňuje propojení objektu FdtChannel s prvky ActiveX.
- **Rozhraní IFdtChannelActiveXControl2**
  Toto nepovinné rozhraní rozšiřuje předcházející rozhraní o nové funkce použitelné ve verzi FDT 1.2.1 a vyšší.

## 5.4. XML schémata FDT

### 5.4.1. XML v FDT

V FDT se téměř pro všechny operace, ať již se jedná o komunikaci, konfiguraci, ukládání dat apod., používá XML dokumentů. Pro každou specifickou operaci je definováno příslušné XML schéma s odpovídajícími atributy a elementy příslušející k dané operaci. Použití konceptu těchto XML dokumentů umožňuje snadněji definovat a spravovat všechna data, která jsou pro běh rámcové aplikace, DTM, BTM či komunikačních kanálů nezbytná.

Detailní popis všech schémat a jejich použití je možné nalézt například v [1].

### 5.4.2. Schéma FDTDataTypesSchema

Schéma datových typů je v FDT použito pro globální definici. Datové typy tohoto schématu jsou v ostatních schématech uváděny předponou „fdt:“.

### 5.4.3. Schéma FDTApplicationIdSchema

Toto schéma určující kontext dané aplikace je v FDT použito pro globální definici. Identifikátory aplikace jsou v ostatních schématech uváděny předponou „fdtappid:“. Vzhled a funkčnost uživatelského rozhraní v DTM jsou kontrolovány elementy applicationId, functionId a operationPhase z tohoto schématu.

### 5.4.4. Schéma FDTUserInformationSchema

Toto schéma využívané v FDT pro globální definici informuje DTM během jeho inicializace o roli a uživatelských právech aktuálního uživatele.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<FDT xmlns="x-schema:FDTUserInformationSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
    <FDTUserInformation userName="ThisUser" userLevel="maintenance" loginTime="2000-05-07T18:39:09"
    loginLocation="WorkStation10" sessionDescription="The actor may specify his session here!"/>
</FDT>
```

Obrázek 12 - Příklad použití FDTUserInformationSchema

### 5.4.5. Schéma DTMInformationSchema

V XML dokumentu DTMInformationSchema jsou obsaženy všechny specifické informace o DTM, jako jsou například podporované sběrnice, informace o verzi a výrobci přístroje, podporovaném jazyku apod.

```xml
<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMInformationSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml" >
<DtmInfo>
    <FDTVersion major="1" minor="2" release="1" build="1"/>
    <fdt:VersionInformation name="myname" vendor="myVendor"  version="1.0" date="2000-08-05"/>
    <DtmSchemaPaths>
        <DtmSchemaPath fdt:busCategory="12345678-ABCD-7890-86E1-00E0987270B9" fdt:path="file://c:/myPath/"/>
    </DtmSchemaPaths>
    <DtmDeviceTypes>
        <fdt:DtmDeviceType>
            <fdt:VersionInformation name="myname" vendor="myVendor"  version="1.0" date="2000-08-05"/>
            <fdt:SupportedLanguages>
                <fdt:LanguageId languageId="1131"/>
            </fdt:SupportedLanguages>
            <fdt:BusCategories>
                <fdt:BusCategory busCategory="036D1499-387B-11D4-86E1-00E0987270B9" busCategoryName="Profibus DP/V1">
                    <fdt:CommunicationTypeEntry communicationType="required"/>
                </fdt:BusCategory>
                <fdt:BusCategory busCategory="036D1498-387B-11D4-86E1-00E0987270B9" busCategoryName="HART">
                    <fdt:CommunicationTypeEntry communicationType="supported"/>
                    <fdt:PhysicalLayer physicalLayer="036D1590-387B-11D4-86E1-00E0987270B9" physicalLayerName="IEC61158-2"/>
                </fdt:BusCategory>
            </fdt:BusCategories>
            <fdt:DeviceIcon path="file://c:/programs/manufacturer/icons/device.ico" />
            <fdt:DeviceIcon deviceGraphicState="device" path="file://c:/dev.ico" />
            <fdt:DeviceIcon deviceGraphicState="diagnosis" path="file://c:/devdiag.ico" />
            <fdt:DtmDocuments>
                <fdt:DtmDocument label="Online Help" help="Help for Dtm" documentClassification="help"
                    documentLanguageId="1033">
                    <fdt:DocumentFile file="c:\mydtm\helpEN.chm"/>
                </fdt:DtmDocument>
                <fdt:DtmDocument label="Online Hilfe" help="Hilfe für DTM" documentClassification="help"
                    documentLanguageId="1031">
                    <fdt:DocumentFile file="c:\mydtm\helpDE.chm"/>
                </fdt:DtmDocument>
            </fdt:DtmDocuments>
            <fdt:DeviceBitmap deviceGraphicState="device" path="file://c:/dev.bmp" />
            <fdt:DeviceBitmap deviceGraphicState="diagnosis" path="file://c:/devdiag.bmp" />
            <fdt:DataSetFormats>
                <fdt:Current dataSetID="036D1497-387B-11D4-86E1-00E0987270B9" description="The data set in encrypted
                    according to the client recommendation"/>
                <fdt:Supported dataSetID="036D1497-387B-11D4-86E1-00E0987270B9" levelOfSupport="full"
                    description="The data set is fully supported"/>
                <fdt:Supported dataSetID="036D1497-387B-11D4-86E1-00E0987270BA" levelOfSupport="full"
                    description="The default value for fault action is not supported  in this version"/>
            </fdt:DataSetFormats>
            <fdt:DeviceTypeVariants>
                <fdt:DeviceTypeVariant deviceTypeVariant="PM255-EX" deviceTypeVariantInfo="PM255-EX Info"
                    orderCode="ExampleOrderCode1"/>
                <fdt:DeviceTypeVariant deviceTypeVariant="PM255-TX" deviceTypeVariantInfo="PM255-TX Info"
                    orderCode="ExampleOrderCode2"/>
            </fdt:DeviceTypeVariants>
        </fdt:DtmDeviceType>
    </DtmDeviceTypes>
</DtmInfo>
</FDT>
```

Obrázek 13 - Příklad XML dokumentu DTMInformationSchema

### 5.4.6. Schéma DTMParameterSchema

XML dokument DTMParameterSchema slouží v FDT k určení specifických informací o zařízení. V tomto dokumentu jsou uloženy informace o adrese zařízení, konfigurační informace vztahující se ke sběrnici, podle kterých potom může rámcová aplikace spravovat příslušnou komunikaci, nebo unikátní identifikátor daného zařízení v systému.

Přistupovat a měnit parametry v tomto dokumentu lze pomocí funkcí z FDT příslušejících rozhraní IDtmParameter – *IDtmParameter::GetParameters()* a *IDtmParameter::SetParameters()*.

### 5.4.7. Schéma DTMFunctionsSchema

V tomto XML dokumentu jsou ve strukturované formě popsány funkce, které jsou nabízeny modulem DTM rámcové aplikaci. Tyto funkce mohou být standardní (tzn. jsou určeny obecným identifikátorem application id) nebo mohou být specifické pro DTM. Kromě identifikátorů jednotlivých funkcí obsahuje tento dokument rovněž informace, jak má rámcová aplikace s těmito funkcemi nakládat. Tedy jestli jsou funkce v dané chvíli spustitelné, zda mají grafické rozhraní nebo běží pouze v těle DTM apod.

Přistupovat k parametrům tohoto dokumentu je možné pomocí funkce *IDtm::GetFunctions()*.

```
<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMFunctionsSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml"
    xmlns:appId="xschema:FDTApplicationIdSchema.xml" defaultFunctionId="1">
    <Functions label="Functions" fdt:name="" help="">
        <StandardFunctions label="Standard Functions" fdt:name="" help="">
            <StandardFunction fdt:name="Observe" help="Help text for Observe" functionId="1">
                <Status toggle="1" checked="0" enabled="1" hidden="0" separator="0"/>
                <appId:ApplicationId applicationId="fdtObserve"/>
            </StandardFunction >
            <StandardFunction fdt:name="Configuration" help="Help text for Configuration" functionId="2">
                <Status toggle="1" checked="0" enabled="1" hidden="0" separator="0"/>
                <appId:ApplicationId applicationId="fdtConfiguration "/>
            </StandardFunction >
        </StandardFunctions >
        <Functions label="Extras" fdt:name="My extra functions" help="Help text for my extra functions">
            <Function label="Extra entry 1" fdt:name="Extra 1" help="Help for extra 1" hasGUI="1" functionId="65"/>
            <Function label="Extra entry 2" fdt:name="Extra 2" help="Help for extra 2" functionId="75"/>
        </Functions>
    </Functions>
</FDT>
```

Obrázek 14 - Příklad XML dokumentu DTMFunctionsSchema

### 5.4.8. Další XML schémata

V předcházejících kapitolách byla popsána ta nejdůležitější rozhraní, s kterými se může uživatel v FDT osobně setkat. Tato schémata však tvoří pouze zlomek ze všech schémat, která se v FDT používají. Celkem je těchto schémat přes čtyřicet a jsou k nalezení v [1].

# 6. PROFIBUS

## 6.1. Úvod

V polovině 80. let 20. století se seskupení firem Siemens, Bosch a dalších rozhodlo vyvinout novou průmyslovou sběrnici PROFIBUS (PROcess FIeld BUS), jejíž zásadními výhodami měla být otevřenost pro každého výrobce, rychlost, univerzálnost a vysoká spolehlivost. Tato sběrnice byla poprvé popsaná v roce 1989 v německé normě DIN 19245. V roce 1996 byla schválena jako evropský standard normou EN 50170. Od roku 2000 je sběrnice PROFIBUS pod označením IEC 61 158 celosvětovým standardem.

## 6.2. Rozdělení PROFIBUS

V současnosti je sběrnice PROFIBUS dle normy rozdělena do tři komunikačních standardů. Každý z těchto standardů má vlastní specifika, kterými se odlišuje od zbývajících dvou standardů. K dispozici jsou tyto tři standardy:

- PROFIBUS FMS (FieldBus Message Specifikation)
- PROFIBUS DP (Decentralized Pheriphery)
- PROFIBUS PA (Process Automation)

## 6.3. PROFIBUS FMS

PROFIBUS FMS je nejstarší variantou standardu sběrnice PROFIBUS je založen na systému komunikačních rámců. Před rychlostí odezvy sběrnice je zde spíše kladen důraz na komfort služeb systému, které jsou zastoupeny množstvím nadstandardních funkcí pro komunikaci mezi inteligentními provozními přístroji. V současné době je však použití sběrnice PROFIBUS FMS na ústupu a je nahrazováno například uživatelsky pohodlnějším Industrial Ethernetem.

## 6.4. PROFIBUS DP

PROFIBUS DP je založen na přenosovém médiu dle standardu RS-485 (kroucená dvoulinka) nebo na optickém vlákně, čímž je zaručena vysoká přenosová rychlost. Přenosová rychlost může v závislosti na rozsáhlosti a složitosti topologie sítě dosáhnout až 12 Mbit/s.

Sběrnicový systém je založen na komunikačním principu master/slave, kde master vystupuje jako aktivní a slave jako pasivní člen. Pokud je na sběrnici připojeno více jednotek typu master, dochází mezi nimi k předávání pověření (tokenu) cyklicky v logickém kruhu. Při odpojení jednotky z kruhu dochází k rekonfiguraci a komunikace může pokračovat dál.

Z důvodu vysoké přenosové rychlosti, vysoké spolehlivosti a značné flexibility je PROFIBUS DP nejrozšířenějším sběrnicovým systémem v Evropě a je využíván ve všech oblastech průmyslové automatizace.

## 6.5. PROFIBUS PA

PROFIBUS PA je nejmladší z rodiny sběrnicových protokolů typu PROFIBUS. Vznikl jako rozšíření varianty DP pro práci v náročném a výbušném prostředí a proto se používá především v oblastech, kde je třeba zajistit jiskrovou bezpečnost Ex (Explosive protection), jako je petrochemický, farmaceutický nebo potravinářský průmysl.

Hlavní rozdíl mezi variantou DP a PA je tvořen nahrazením analogového přenosu měřené veličiny její digitální formou, která může být ještě před odesláním do řídícího systému předzpracována a upravena. Fyzické médium, po kterém jsou informace přenášeny, je normou označeno jako IEC 1158-2 (na rozdíl od RS-485, která je používána u PROFIBUS DP) a kromě komunikace, jejíž rychlost je u PROFIBUSu PA omezena na maximálně 93,75 kbit/s, slouží rovněž k napájení provozního přístroje.

Oba typy sběrnice, PROFIBUS PA a DP, je mezi sebou možno vzájemně propojit. Ke spojení se používá dvou typů propojovacích členů. První se označuje jako vazební člen DP/PA (DP/PA link) a jeho výhodou je, že rychlost PROFIBUSu DP nemusí být omezována na rychlost PROFIBUSu PA. Nevýhodou je, že tento vazební člen musí být zahrnut do hardwarové konfigurace. Druhou, jednodušší možností je použití převodníku DP/PA (DP/PA coupler), který se do hardwarové konfigurace nijak nepromítne. Nevýhodou použití převodníku je, že je nutné snížit přenosovou rychlost PROFIBUSu DP na rychlost PROFIBUSu PA.

PROFIBUS PA je podobně jako PROFIBUS DP založen na komunikačním principu master/slave pojmenovaném jako DPV1. V praxi mohou na sběrnici přistupovat tři typy zařízení:

- **Master třídy 1 (DPMC1)**
  Master třídy 1 realizuje cyklickou komunikaci se zařízeními typu slave a v praxi je většinou tvořeno programovatelným automatem PLC nebo CNC.
- **Master třídy 2 (DPMC2)**
  Master třídy 2 je používán pro acyklickou komunikaci a to především při konfiguraci, diagnostice nebo dohledu nad řídícím systémem. V praxi je master třídy 2 většinou realizován pomocí PC (rámcová aplikace FDT spolu s DTM a komunikační kartou tvoří tedy master třídy 2).
- **Slave**
  Slave představuje pasivní periferní člen sloužící ke sběru dat nebo vykonávání akčních zásahů.

## 6.6. Profily PROFIBUS PA

### 6.6.1. Úvod

K vytvoření profilů PROFIBUS PA vedla snaha standardizovat provozní přístroje a vytvořit skupiny zařízení s podobným chováním, aby bylo možné mezi různými typy zařízení z dané skupiny nezávisle na výrobci volit a tato zařízení volně zaměňovat. Pro každou skupinu přístrojů se specifickou funkcí byl definován jeden profil, který definuje chování zařízení či jeho data. Profily zařízení PROFIBUS PA jsou vydávány německou organizací PNO (Profibus NutzerOrganisation) a v současné době existují ve verzi 3.0.

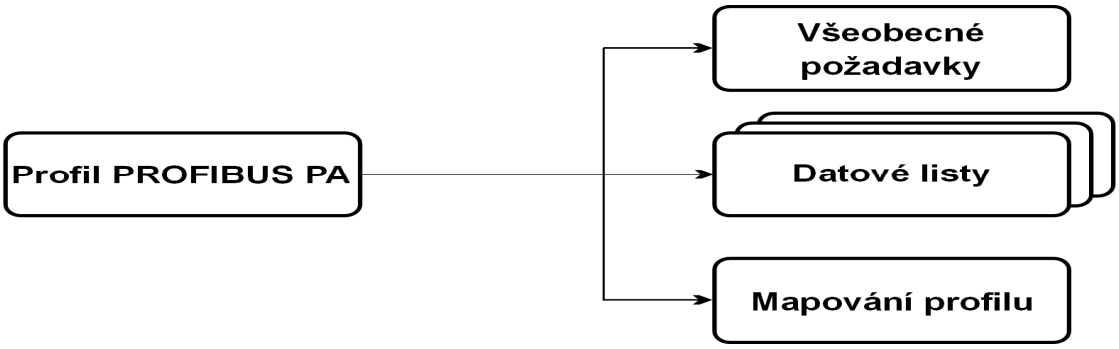
### 6.6.2. Rozdělení profilu PROFIBUS PA

Profil PROFIBUS PA je rozdělen na tři části. První část je tvořena všeobecnými požadavky (General Requirements). V těchto požadavcích jsou popsána všeobecná definice a pravidla pro všechna zařízení pracující na tomto profilu. Profil rozděluje zařízení dle těchto všeobecných požadavků na dvě třídy:

- **Třída A** (class A) popisuje parametry pro jednoduchá zařízení, která poskytují pouze základní funkce.
- **Třída B** (class B) rozšiřuje možnosti třídy A o parametry pro identifikaci, údržbu a diagnostiku.

Druhou část profilu PROFIBUS PA tvoří datové listy (Data Sheets) zařízení. Tyto datové listy obsahují popis fyzického, převodního a funkčního bloku.

Třetí část struktury profilu tvoří mapování profilu (Mapping to fieldbus) na služby sběrnice PROFIBUS DP.




Obrázek 15 - Rozdělení profilu PROFIBUS PA

### 6.6.3. Struktura profilu PROFIBUS PA

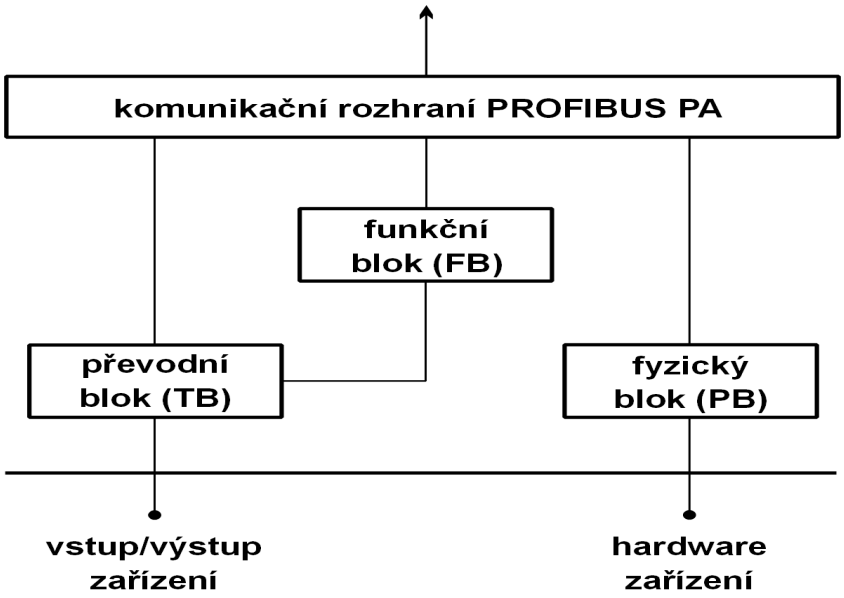
Vnitřní struktura zařízení pracujícím na profilu PROFIBUS PA je složena ze specializovaných bloků, jenž každý plní svoji specifickou úlohu v celém systému. Model zařízení je dvojího druhu a liší se podle toho, zda se jedná o kompaktní nebo modulární zařízení.

Kompaktní zařízení jsou jednoduchá zařízení, u kterých se nepředpokládá zpracování měřených hodnot více různými způsoby. Kompaktní zařízení obsahuje vždy jeden fyzický blok (Physical Block – PB), nejméně jeden převodní blok (Transducer Block – TB) a nejméně jeden funkční blok (Function Block – FB). Tyto bloky jsou zastřešeny blokem správce zařízení (Device Management – DM).

Modulární zařízení jsou zařízení složitější konstrukce složené z více modulů. Nad těmito moduly stojí jeden základní DM a PB, které jsou společné pro všechny moduly uvnitř modulárního zařízení. V každém modulu je pak jeden PB příslušející tomuto modulu a nejméně jeden TB a FB podobně jako u

kompaktního zařízení. Kromě těchto bloků obsahuje modulární zařízení ještě spojovací objekt (Link Object), který slouží k vzájemnému propojení funkčních bloků jednotlivých modulů.

Ať již se jedná o kompaktní nebo modulární zařízení, správce zařízení (DM) je nezávisle na typu zařízení vždy umístěn stejně. V DM jsou umístěny odkazy na jednotlivé fyzické, převodní a funkční bloky (popřípadě i na spojovací objekt) a jejich počet. Ve fyzickém bloku zařízení (PB) jsou většinou obsaženy statické informace o zařízení a hardwaru. Převodní blok (TB) zajišťuje spojení s měřícím čidlem nebo akčním členem a dochází v něm ke korekci naměřených hodnot, převodu na různé fyzikální jednotky, linearizaci apod. Funkční blok (FB) dále zpracovává informace přijaté z převodního bloku a provádí s nimi konečnou úpravu.



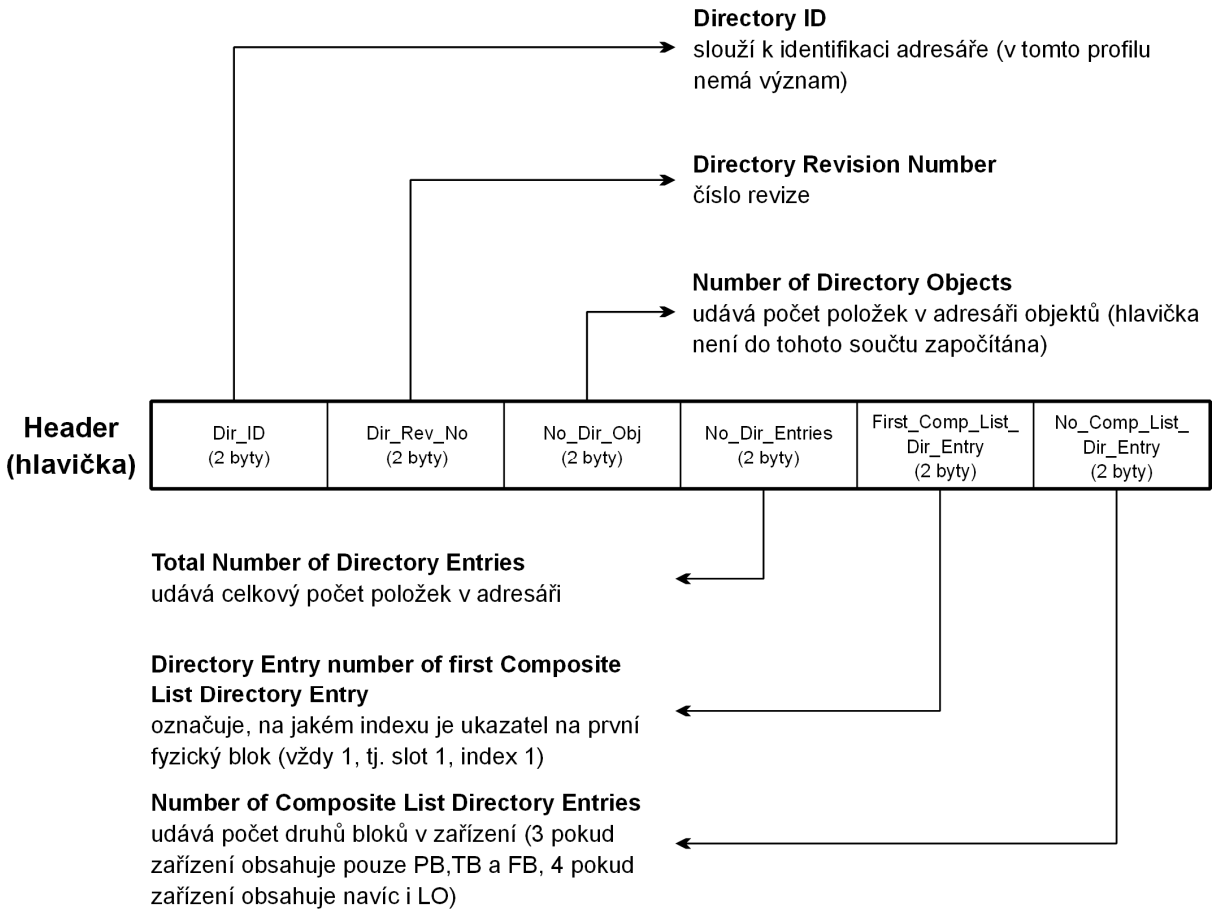

Obrázek 16 - Struktura profilu PROFIBUS PA

### 6.6.4. Správce zařízení a adresář objektů

Nejdůležitější součástí správce zařízení (DM) je adresář objektů (Directory Object). V adresáři objektů jsou uloženy pozice jednotlivých bloků zařízení. Pro každý blok je rovněž možné zjistit jeho typ, adresu první proměnné daného bloku a počet všech parametrů v bloku obsažených. Norma určuje, že počátek

adresáře objektů se vždy nachází na pevně daném umístění – slot 1, index 0. Adresář objektů se skládá ze tří důležitých položek:

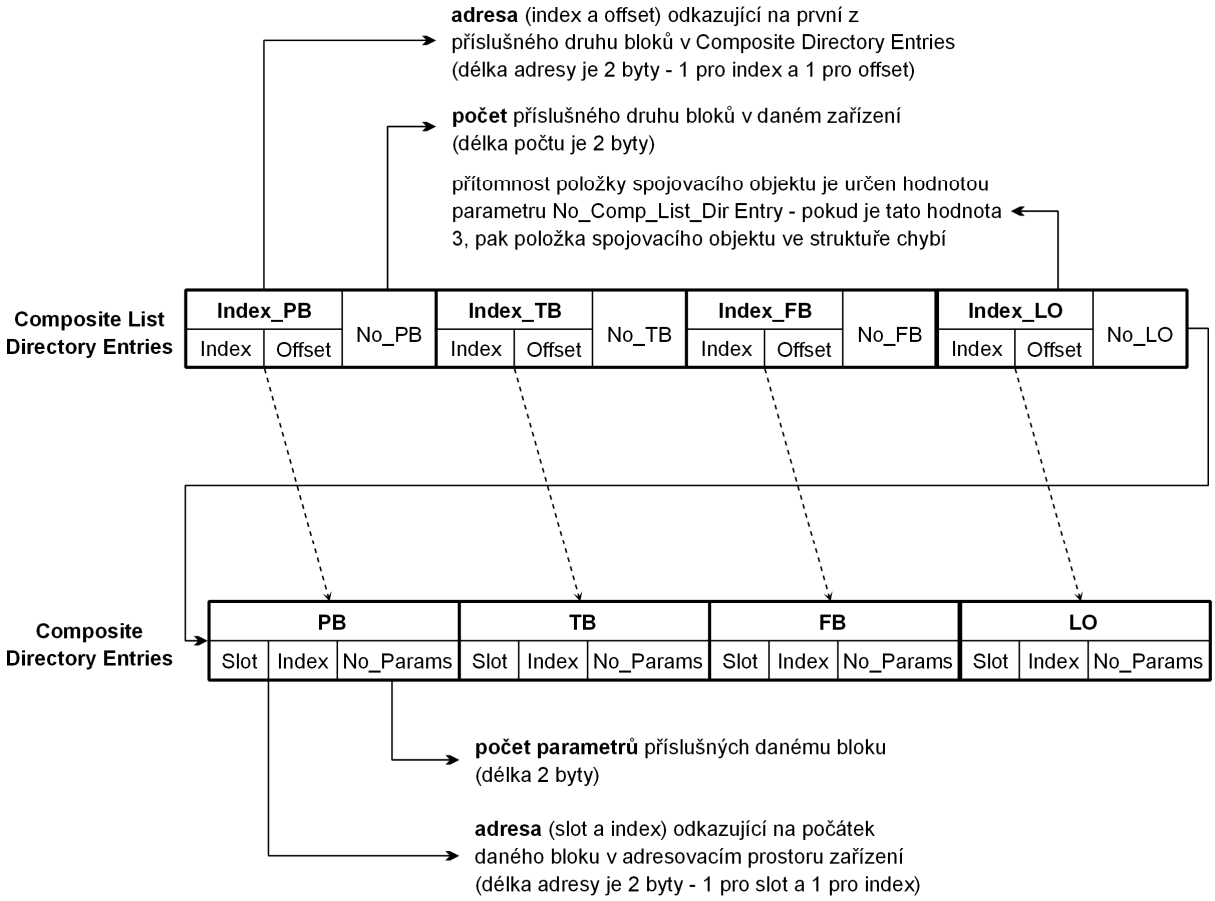
- **Header** (hlavička) – V hlavičce adresáře objektů se nacházejí informace o struktuře a počtu objektů v adresáři objektů.




Obrázek 17 - Struktura hlavičky adresáře objektů

- **Composite List Directory Entries** (souhrnné informace) – Položka souhrnných informací obsahuje informace o blocích obsažených v daném zařízení a je vždy adresována na pozici slot 1, index 1. Podle hodnoty No_Comp_List_Dir_Entry uvedené v hlavičce mohou mít souhrnné informace tři nebo čtyři položky, pro každý druh bloku přítomného v zařízení jednu. Jednotlivé položky odkazují pro daný druh bloku do položky Composite Directory Entries, kde jsou přesnější informace o blocích tohoto typu. Kromě toho informují souhrnné informace o počtu bloků každého druhu.

- **Composite Directory Entries** (ukazatele na bloky) – Tato položka adresářového objektu dynamicky navazuje na položku souhrnných informací. V této položce jsou zaznamenány ukazatele (slot a index) na konkrétní bloky a počet parametrů obsažených v těchto konkrétních blocích. Pro každý blok v zařízení zde existuje samostatná položka a položky bloků stejného typu jsou řazeny za sebou (ukazatele z Composite List Directory Entries směřují vždy na první z těchto položek).




Obrázek 18 - Vnitřní struktura v adresáři objektů

### 6.6.5. Parametry bloků

Každý blok obsažený v provozním přístroji a odpovídající profilu PROFIBUS PA obsahuje standardní a rozšiřující parametry. Standardní parametry jsou pro všechny bloky stejné a musí být v daném bloku uvedeny – jsou povinné.

Ze standardních parametrů je nejdůležitější parametr zvaný **Block Object**. Tento parametr obsahuje charakteristiku celého bloku a jeho vnitřní strukturu můžeme vidět v Tabulce 1.

| E | Element Name | Data Type | (Index) | Size |
|---|---|---|---|---|
| 1 | Reserved | Unsigned8 | (5) | 1 |
| 2 | Block_Object | Unsigned8 | (5) | 1 |
| 3 | Parent_Class | Unsigned8 | (5) | 1 |
| 4 | Class | Unsigned8 | (5) | 1 |
| 5 | DD_Reference | Unsigned32 | (7) | 4 |
| 6 | DD_Revision | Unsigned16 | (6) | 2 |
| 7 | Profile | OctetString | (10) | 2 |
| 8 | Profile_Revision | Unsigned16 | (6) | 2 |
| 9 | Execution_Time | Unsigned8 | (5) | 1 |
| 10 | Number_of_Parameters | Unsigned16 | (6) | 2 |
| 11 | Address_of_View_1 | Unsigned16 | (6) | 2 |
| 12 | Number_of_Views [1] | Unsigned8 | (5) | 1 |

Tabulka 1 - Struktura parametru Block Object

Podle vnitřních parametrů Block Objectu lze rozlišit, jakého je daný blok typu (parametr Block_Object), jakou funkci daný blok vykonává (Parent_Class), jakou má specializaci (Class) nebo jaká je verze profilu na které zařízení pracuje (Profile_Revision). Příklad pro první čtyři byty Block Objectu fyzického bloku je vidět v Tabulce 2.

| Byte 1 | Byte 2 | Byte 3 | Byte 4 |
|---|---|---|---|
| **Reserved** | **Block_Object** | **Parent_Class** | **Class** |
| 0 – 127 reserved<br>Default = 250 (not used) | 01 Physical Block | 01 Transmitter<br>02 Actuator<br>03 Discrete I/O<br>04 Controller<br>05 Analyser<br>06 Lab Device<br>07 reserved<br>...<br>126 reserved<br>127 Multi-Variable | Default = 250 (not used) |

Tabulka 2 - Block Object pro fyzický blok

Rozšiřující parametry jsou závislé jednak na typu zařízení a jednak na druhu bloku. Výjimku v tomto případě tvoří fyzický blok, jehož rozšiřující parametry jsou pro všechna zařízení stejné, nezávisle na jejich typu. Norma rozděluje zařízení do těchto typů:

- digitální vstupy a výstupy
- snímače (tlaku, teploty, hladiny atd.)
- akční členy (elektropneumatické, elektrické atd.)
- analyzátory (analogové, digitální (I/O)
- konfigurovatelné (kombinace předchozích typů)

Rozšiřující parametry mohou být povinné (mandatory), které musí být v zařízení uvedeny, a nepovinné (optional), u nichž záleží na výrobci, zda je do svého zařízení zahrne.

Protože zařízení, se kterým se v této diplomové práci pracuje, je teplotní snímač, budou všechny parametry, o kterých se dále bude mluvit, specifické pro skupinu snímačů teploty. Jednotlivé parametry pro fyzický, převodní a funkční blok jsou blíže popsány v příloze nebo v [2].

# 7. Vývoj softwarových modulů DTM

## 7.1. Úvod

Základním stavebním kamenem standardu FDT (Field Device Tool) je modul DTM (Device Type Manager). Každé zařízení připojené do řídícího systému potřebuje svůj vlastní ovladač DTM, který je obvykle dodáván výrobcem spolu se zařízením. Výhodou takovéto struktury je, že rozsáhlý řídící systém může být snadno dynamicky měněn či rozšiřován instalací nového ovladače DTM.

V praxi se moduly DTM rozdělují do dvou kategorií – ovladač zařízení DeviceDTM a komunikační ovladač CommDTM. DeviceDTM v sobě uchovává všechna data a informace o zařízení nebo skupině zařízení, pro která je ovladač určen. Umožňuje ovládat příslušné zařízení a obsahuje funkce pro jeho obsluhu, jež jsou přístupné skrze prostředí rámcové aplikace. Komunikační ovladač CommDTM představuje jakéhosi prostředníka mezi DeviceDTM a fyzickou vrstvou, na které je zařízení umístěno. Zajišťuje obsluhu komunikačního kanálu a komunikačních procedur, které jsou nezbytné pro správnou funkci zařízení.

Pro splnění stanovených cílů diplomové práce musíme vytvořit jak DeviceDTM, který bude obsahovat ovládací funkce pro teploměr TF-211 společnosti ABB, tak i CommDTM, který nám umožní připojit se do sítě Profibus PA a komunikovat se zařízením podle protokolu DPV1 přes softwarovou kartu CP-5611 společnosti Siemens.

## 7.2. Prostředky pro vývoj DTM

Protože vývoj DTM bez podpůrných prostředků by byl velice náročný, existují pomocné nástroje, které návrh a vývoj ovladačů DTM usnadňují. Ze všech existujících nástrojů jmenujme například DTMCreator Base společnosti iFak System GmbH nebo dtmManager společnosti M&M Software.

### 7.2.1. DTMCreator Base (iFak System GmbH)

DTMCreator Base je vývojový nástroj, který usnadňuje vývoj ovladače DTM, neboť redukuje čas a cenu vývojového procesu.

DTMCreator Base obsahuje následující komponenty:

- DTM Wizard pro návrh modulů DTM
- knihovna šablon
- knihovna objektů
- FDT MiniFrame (rámcová aplikace)

DTMCreator Base pomáhá při vývoji komunikačních modulů CommDTM a modulů pro zařízení DeviceDTM, podporuje komunikační protokoly HART, Profibus, Foundation Fieldbus a rovněž podporuje OPC. Při vývoji provází DTM Wizard krok za krokem skrz návrh DTM ve Visual C++. Uživatel je veden dialogy mezi různými typy DTM nebo podporovanými protokoly. Ve výsledku je vygenerován projekt Visual C++, který může být zkompilován do vyhovujícího DTM. DTM Wizard vytvoří rámcový projekt pro DTM, který obsahuje množství ActiveX funkcí a komunikačních kanálů pro CommDTM.

### 7.2.2. dtmManager 2.1 (M&M Software GmbH)

dtmManager 2.1 je robustní nástroj pro Visual C++ určený pro vývoj CommDTM a DeviceDTM podle FDT specifikace ve verzi 1.2. Jméno knihovny použité pro tvorbu projektu je DTMBaseLibrary. dtmManager nabízí tyto součásti:

- knihovnu tříd ve Visual C++
- podporu pro tvorbu CommDTM a DeviceDTM
- implementaci nezbytných rozhraní na základě šablon
- robustní implementaci DTM State Machine
- koncepci callbacků pro maximální flexibilitu
- vestavěnou podporu pro XML dokumenty FDT – *DTMInformation*, *DTMParameter* a *DTMFunctions*
- správu perzistentních dat pomocí Persistence Manageru
- podporu pro ukládání logovacích dat
- podporu pro vícejazyčnou lokalizaci textů
- vestavěnou podporu FDT komunikačních kanálů
- tutoriál obsahující příklady pro tvorbu DTM pro zařízení pracující na protokolu PROFIBUS DPV1, příklady implementace komunikace po PROFIBUS PA a příklady pro tvorbu CommDTM

### 7.2.3. Volba vývojového nástroje

Přestože DTMCreator Base je pro tvorbu DTM velice mocný nástroj, je jeho robustnost a všestrannost vykoupena vysokou pořizovací cenou. Z hlediska našich potřeb a cílů se tak ve výsledku jeví jako výhodnější použití dtmManageru, neboť splňuje všechny požadavky, které na vytvářený DeviceDTM a CommDTM klademe. Tento nástroj obsahuje podporu pro komunikační protokol PROFIBUS PA, na kterém funguje i naše zařízení. Kromě toho umožňuje větší svobodu při návrhu, neboť uživatel není omezován žádnými dialogy a může vše vytvořit podle svých představ a požadavků (což je však zároveň i nevýhoda, neboť je od uživatele předpokládána alespoň rámcová znalost principů FDT). Velkou výhodou jsou rovněž návodové příklady, které umožní uživateli lépe pochopit principy, na kterých FDT funguje, a snáze tak proniknout do samotné tvorby DTM modulů.

Pro účely diplomové práce propůjčila katedře Řídící techniky společnost M&M Software svůj produkt dtmManager bezplatně.

## 7.3. Popis vývojového nástroje dtmManager 2.1

Jak již bylo řečeno v předchozí kapitole, dtmManager obsahuje výukový tutoriál, který vysvětluje, jak správně navrhnout modul DeviceDTM či CommDTM, několik příkladů zdrojových kódu usnadňujících pochopení tvorby DTM, a především pak DTMBaseLibrary.

DTMBaseLibrary je ve své podstatě soubor knihoven pro Visual C++, které v sobě obsahují funkce pro vytvoření modulu DTM na základě standardu FDT a které jsou pro funkčnost vytvářeného DTM nezbytné.

V průběhu tvorby CommDTM a DeviceDTM došlo k získání nové verze dtmManageru – verze 2.3. Tato verze je v jádru velice podobná popisované verzi 2.1. Veškeré odlišnosti, kterými jsou obě verze rozdílné, budou v dalším textu zmíněny.

### 7.3.1. Statické a dynamické knihovny

Základem DTMBaseLibrary je statická knihovna *DTMBaseLibrary.lib*. Tato knihovna obsahuje podporu pro většinu funkcí a rozhraní, které DTM potřebuje

ke svému bezchybnému běhu. Místo *DTMBaseLibrary.lib* můžeme v projektu použít knihovnu *DTMBaseLibraryUnicode.lib*, která je co do funkčnosti se základní knihovnou shodná, avšak navíc v sobě obsahuje podporu pro UNICODE.

Pro podporu rozhraní definovaných standardem FDT je zde obsažena dynamická knihovna *DTMBaseLibraryInterfaces.dll*. Samotná rozhraní a funkce pro FDT ve verzi 1.2.1 jsou umístěna v dynamické knihovně *FDT100.dll*. Přestože je tato knihovna vytvořena pro specifikaci FDT ve verzi 1.2.1, zbývající části dtmManageru 2.1 podporují pouze verzi 1.2.0, takže funkce nových rozhraní není možné použít. Konkrétně se jedná o rozhraní *IDtm2*, *IDtmEvents2*, *IDtmInformation2*, *IFdtActiveX2*, *IFdtCommunicationEvents2*, *IFdtChannel-SubTopology2*, *IFdtChannelActiveXControl2*. Verze dtmManageru 2.3 již plně podporuje verzi FDT 1.2.1 a je již možné používat všechna rozhraní bez omezení.

Pro usnadnění vytváření grafických GUI (Graphic User Interface) jsou do DTMBaseLibrary umístěny dvě dynamické knihovny *DTMBaseLibrary-Resources.dll* a *DTMBaseLibraryUtilities.dll*. Ve verzi 2.3 je knihovna *DTMBaseLibraryUtilities.dll* nahrazena novější knihovnou *dtmMANAGER-Satellite_01.dll*, která uživateli při návrhu nabízí nové možnosti.

Protože celé FDT/DTM je postaveno na XML (eXtended Markup Language), DTMBaseLibrary obsahuje dynamickou knihovnu *msxml3.dll*, která zajišťuje podporu pro správu těchto struktur.

Poslední skupinou knihoven jsou statické knihovny, které jsou nezbytné pro úspěšné slinkování projektu DTM ve Visual C++. Knihovny jsou rozděleny podle toho, pro kterou verzi Visual C++ jsou určeny (verze 6.0, 7.0, 7.1, 8.0, nižší verze nejsou podporovány), zda je projekt kompilován jako UNICODE nebo ANSI a jestli je v možnostech projektu Visual C++ nastaven multithreadový debug dynamických knihoven.

### 7.3.2. Hlavičkové soubory

dtmManager 2.1 obsahuje množství přiložených hlavičkových souborů. Tyto soubory v sobě obsahují deklarace tříd, funkcí a proměnných, od kterých uživatelem vytvářený projekt dědí své vlastnosti, a jsou při kompilaci

přilinkovány k výslednému projektu. Funkce obsažené v těchto souborech jsou specializovány na tvorbu modulů DTM a nabízejí programátorovi široké spektrum využití – od funkcí, které jsou pro funkční chod DTM nezbytné až po funkce, které pouze zvyšují a zlepšují uživatelský komfort.

Základním souborem, který je třeba přilinkovat do projektu, je hlavičkový soubor *DTMBaseLibrary.h*. V tomto souboru jsou importovány základní knihovny potřebné pro DTM, obsahuje rozhodovací makra pro import knihoven potřebných ke kompilaci podle parametrů nastavených v projektu Visual C++ a přilinkovává do projektu většinu dalších hlavičkových souborů.

### 7.3.3. Soubory se šablonami

Soubory se šablonami jsou textové soubory, které obsahují šablony upravených zdrojových kódů. Ve většině případů se jedná o upravená volání funkcí základních standardních rozhraní FDT, která je nutno po implementaci do projektu z přiložených souborů přepsat.

### 7.3.4. Soubory pro tvorbu GUI

Soubory pro podporu tvorby grafického rozhraní obsahují hlavičkové soubory (**.h*) a soubory zdrojového kódu (**.cpp*). V hlavičkových souborech jsou umístěny deklarace tříd, funkcí a proměnných, které jsou uživatelem používány pro tvorbě dialogového rozhraní. V souborech zdrojového kódu je pak umístěna implementace výkonného kódu tříd a funkcí.

Nejdůležitějším souborem z této kategorie je soubor *DTMBaseLibraryUI.h*. V tomto souboru dochází k importu všech důležitých knihoven, které jsou pro tvorbu grafického rozhraní důležité, a přilinkování dalších důležitých souborů z této oblasti. Přestože schopný návrhář by tyto soubory vůbec nemusel využívat, v praxi velice usnadňují tvorbu grafického rozhraní, protože jsou zde definovány grafické komponenty přímo využívající možnosti DTM a funkcí DTMBaseLibrary.

Dalším důležitým souborem z této oblasti je soubor *DTMBaseLibrary.rc*. Jedná se o resource script, který uživatel vkládá do svého projektu, a jsou v něm definovány základní vlastnosti grafických komponent obsažených v DTMBaseLibrary.

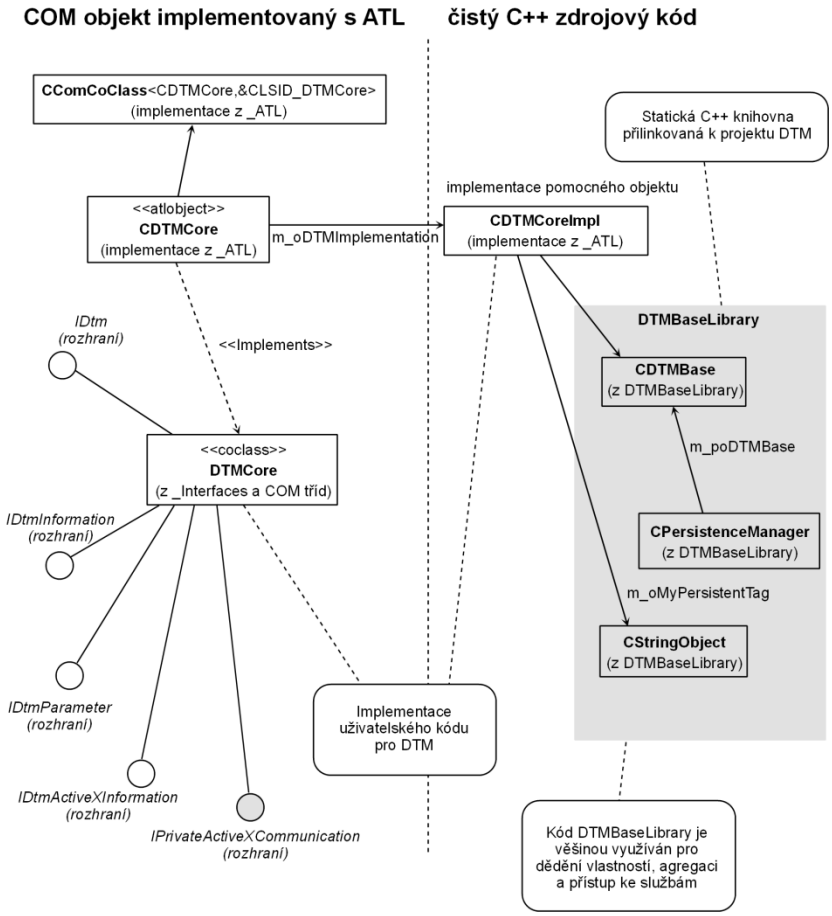
Nezbytnou součástí, která je pro vytváření grafických rozhraní s pomocí DTMBaseLibrary třeba, je grafická knihovna WTL (Windows Template Library), která je jako CPL (Common Public License) zdarma ke stažení například na adrese http://sourceforge.net/projects/wtl.

### 7.3.5. Základní architektura projektu postaveném na DTMBaseLibrary

Základní filozofií projektu vytvořeném na základě DTM Base Library je rozdělení projektu na dvě části – jedna obsahuje ATL COM objekty a druhá čistý C++ zdrojový kód.

V části ATL COM objektu je vytvořena třída CDTMCore, V této třídě se registrují a implementují základní standardní rozhraní FDT.

Z hlediska návrháře je důležitější část čistého C++ zdrojového kódu, která je představována třídou CDTMCoreImpl, která ve své základní podobě dědí vlastnosti od třídy CDTMBase. Ve třídě CDTMCoreImpl jsou implementovány všechny funkce tvořící funkční „tělo“ modulu DTM.

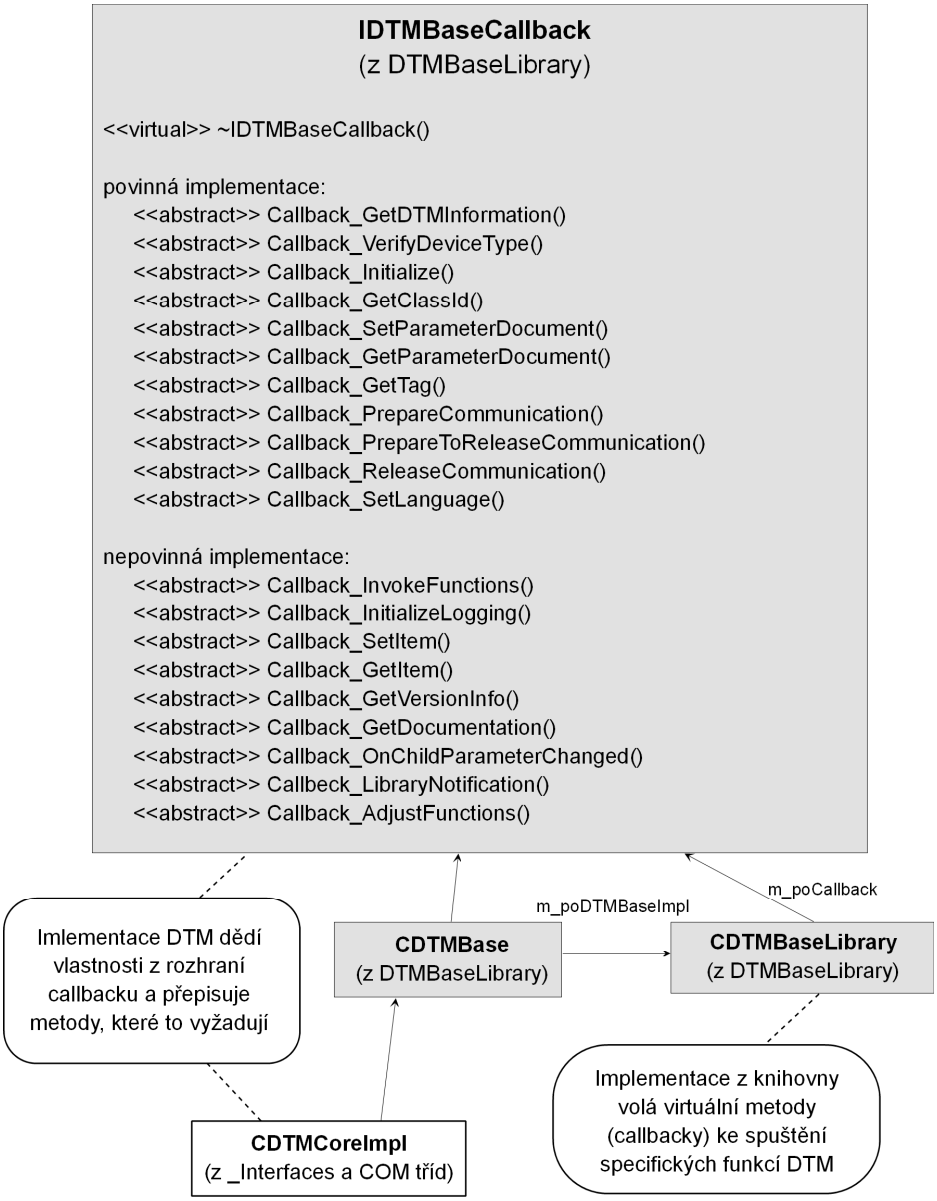



Obrázek 19 - Základní architektura projektu v dtmManageru

### 7.3.6. Koncepce callbacků

Funkční jádro projektu využívající DTMBaseLibrary je založené na koncepci callbacků. Callbacky jsou virtuální metody, které reagují na události probíhající v rámcové aplikaci a obsluhují je. Implementací těl těchto metod může návrhář docílit individuálního chování pro každý specifický modul DTM.

V DTMBaseLibrary existuje dvacet různých callbackových funkcí. Jedenáct z nich je povinných a návrhář je musí do svého projektu implementovat. Zbývajících devět callbacků je nepovinných a programátor je nemusí do svého projektu nezbytně implementovat.



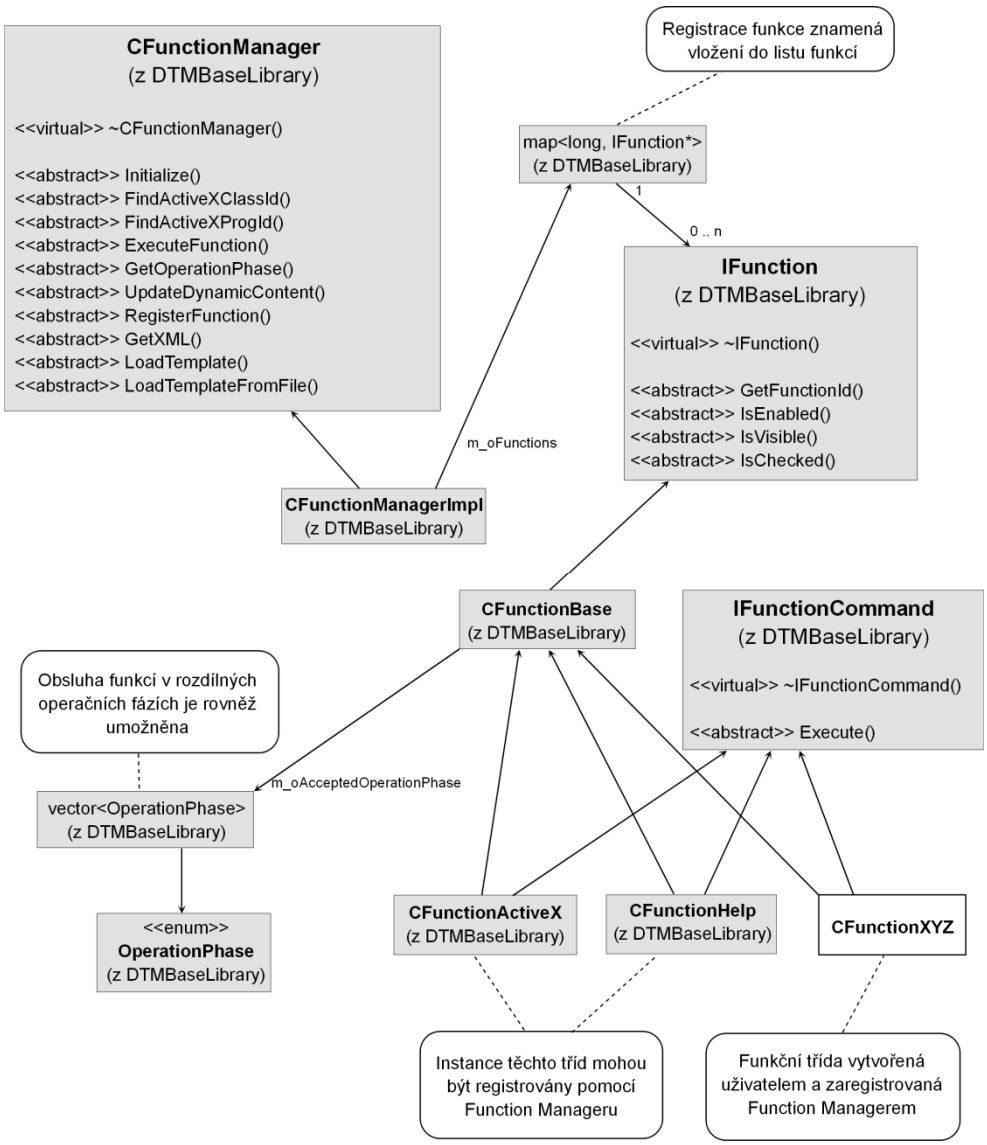

Obrázek 20 - Koncepce callbacků

### 7.3.7. Objekt obsluhy funkcí - Function Manager

Function Manager je objekt DTMBaseLibrary, jenž zapouzdřuje funkce pro obsluhu, správu a registraci funkcí vytvořených uživatelem.

V kontextu rozhraní rámcové aplikace jsou vlastnosti funkcí definovány v dokumentu XML. Pomocí tohoto dokumentu lze určit jméno funkce, její identifikační kód pro jednoznačnou identifikaci, to, zda umožňuje spuštění grafického dialogového rozhraní apod.

Programátorem vytvořený XML dokument, který je specifický pro každé DTM, může být obsažen buď přímo ve zdrojovém kódu projektu, nebo je ho možné načítat z externího souboru. V obou případech se tento dokument načítá při inicializaci modulu DTM, tedy při volání funkce *Callback_Initialize()*.

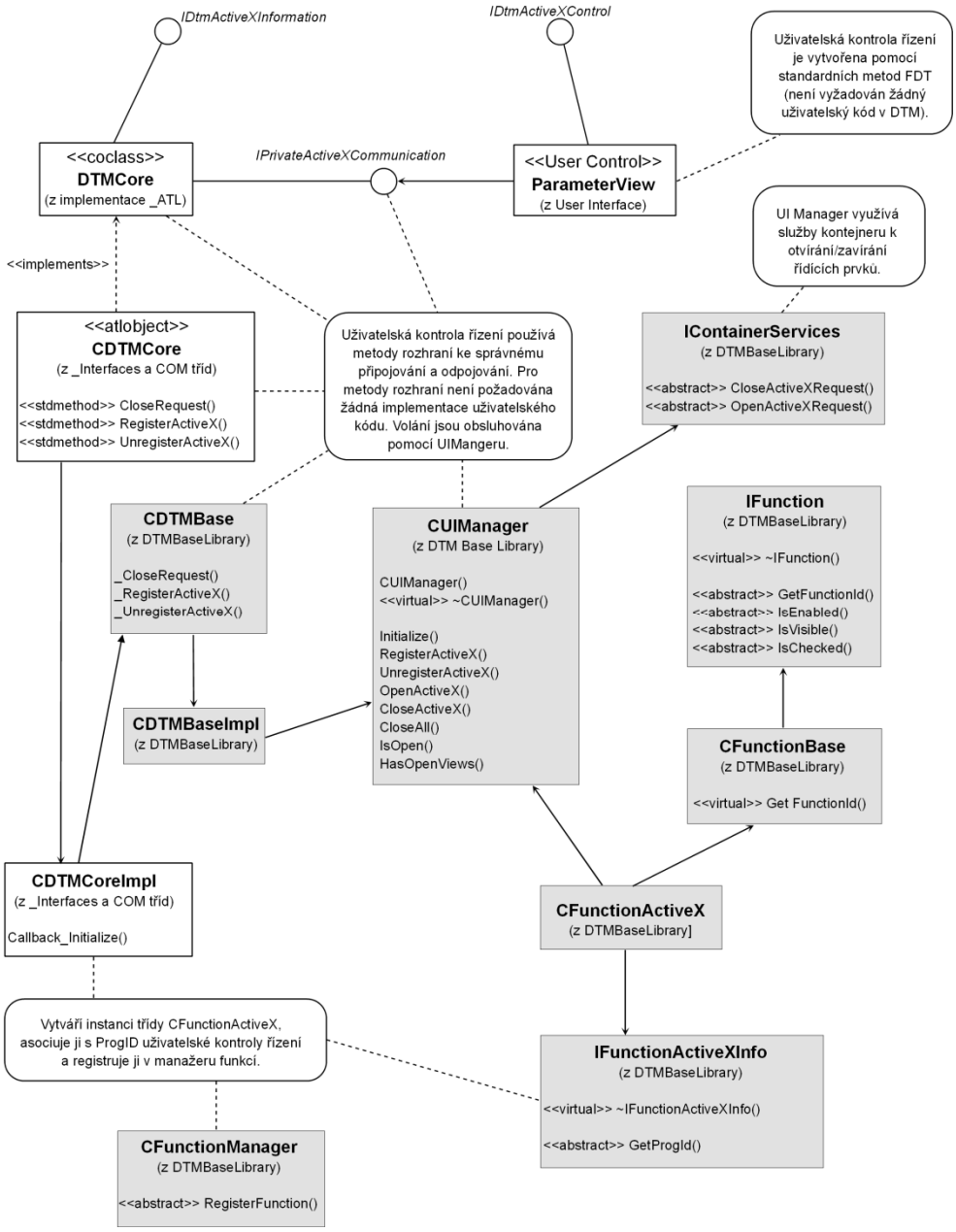



Obrázek 21 - Princip funkce Function Manageru

### 7.3.8. Objekt obsluhy uživatelského rozhraní - UI Manager

Aby operátor mohl pracovat s modulem DTM a přistupovat skrze něj k fyzickému zařízení, je třeba grafického rozhraní, přes které operátor se zařízením komunikuje. Grafické rozhraní v FDT/DTM je založeno na funkcích ActiveX a v DTMBaseLibrary je obsluha těchto funkcí zapouzdřena v třídě CUIManager.

UI Manager (User Interface Manager) zajišťuje především inicializaci a registraci použitých prvků ActiveX a dále pak jejich správu – otvírání a zavírání.

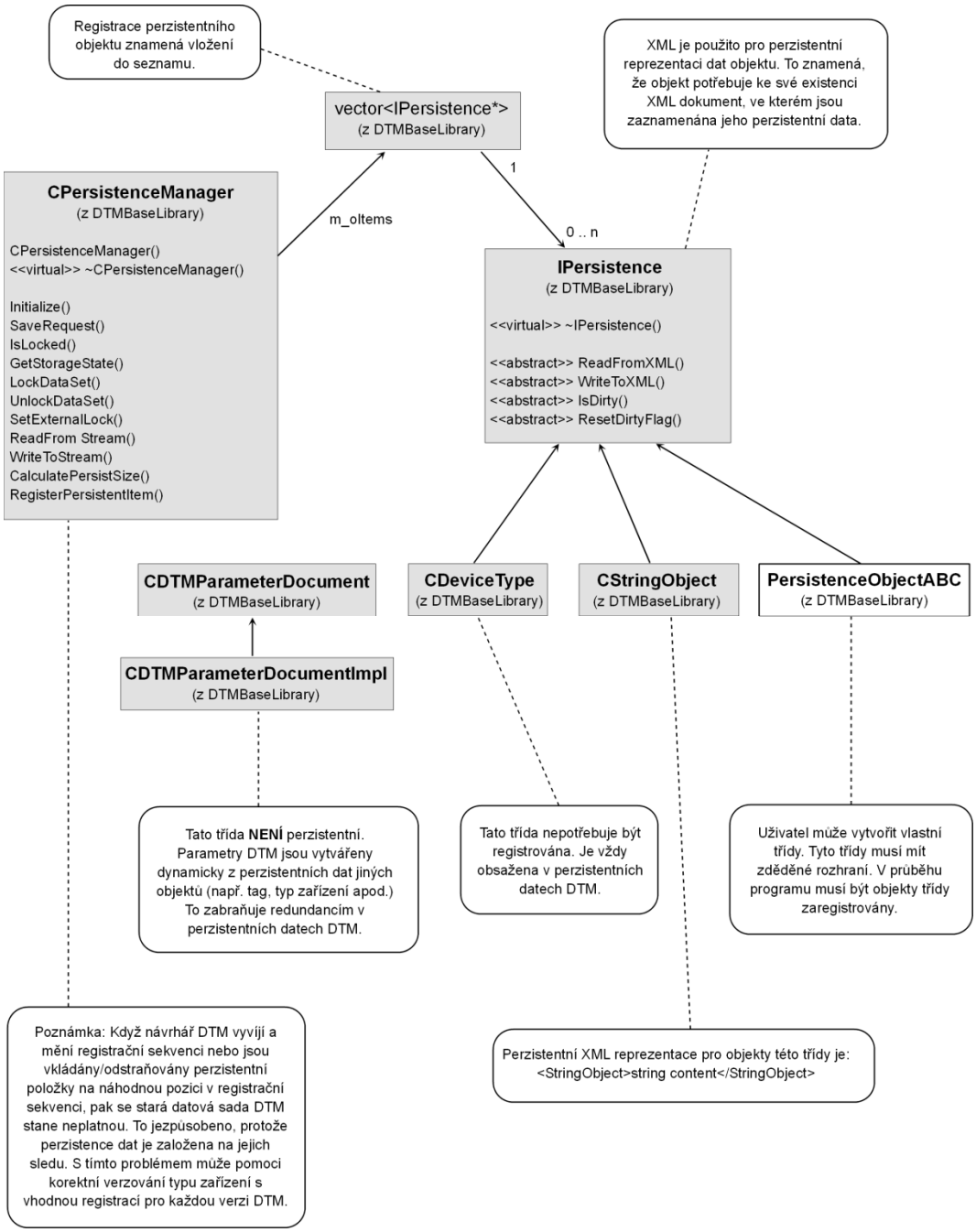



Obrázek 22 - Použití UI Manageru

### 7.3.9. Objekt obsluhy perzistentních dat – Persistence Manager

Pro reprezentaci perzistentních (trvalých) dat používá standard FDT/DTM XML dokumentů. V případě DTMBaseLibrary se o správu, manipulaci a uchovávání perzistentních dat stará objekt obsluhy Persistence Manager.

Persistence Manager pomáhá registrovat jednotlivé perzistentní objekty do přehledového seznamu a udržovat konzistenci na daty.

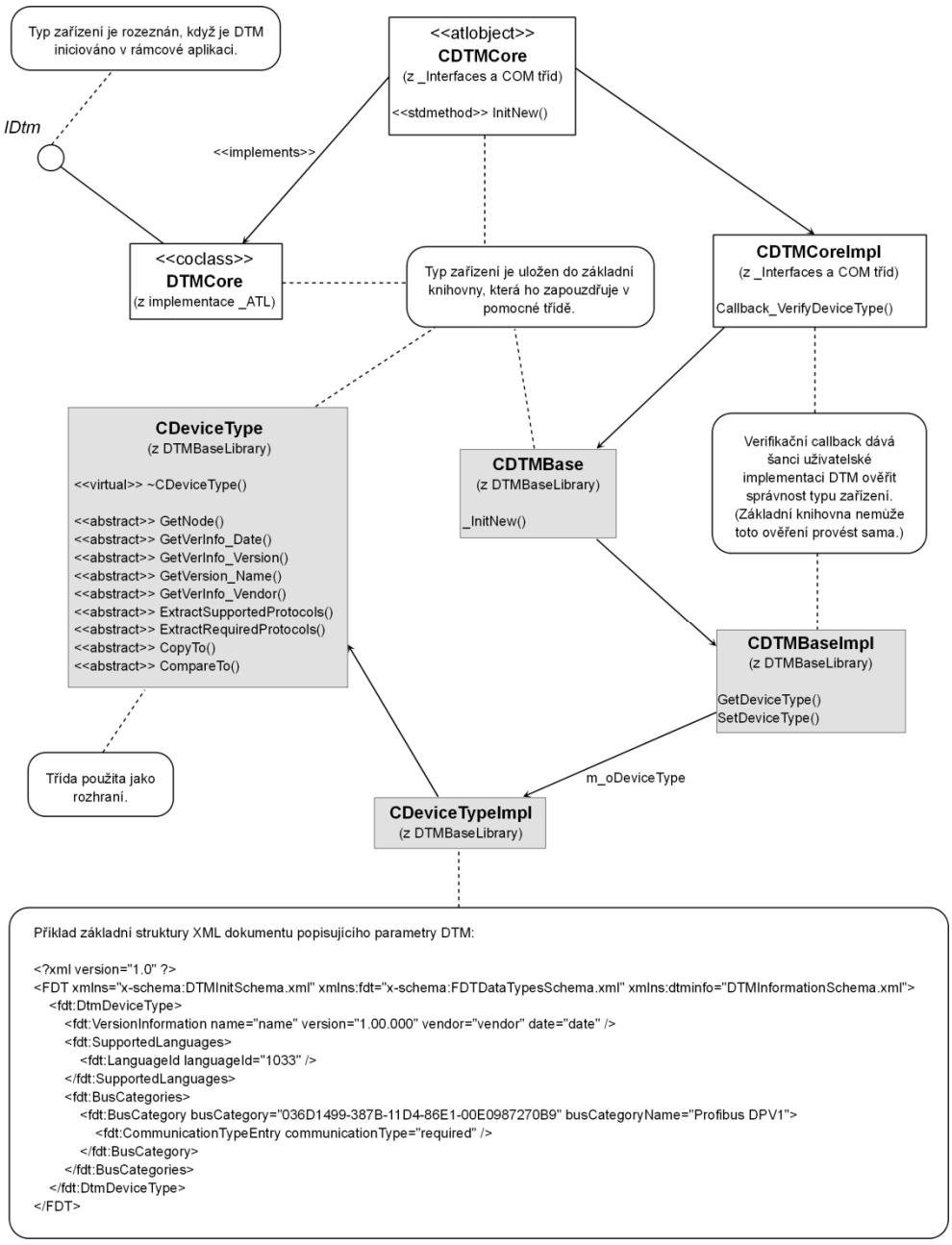



Obrázek 23 - Persistence Manager

### 7.3.10. Obsluha typu zařízení

Třídy pro obsluhu typu zařízení slouží k verifikaci DTM v rámcové aplikaci a k určení, zda je inicializované DTM opravdu určeno pro daný typ zařízení. Tím je zabráněno použití daného DTM pro nevhodné a neodpovídající zařízení.

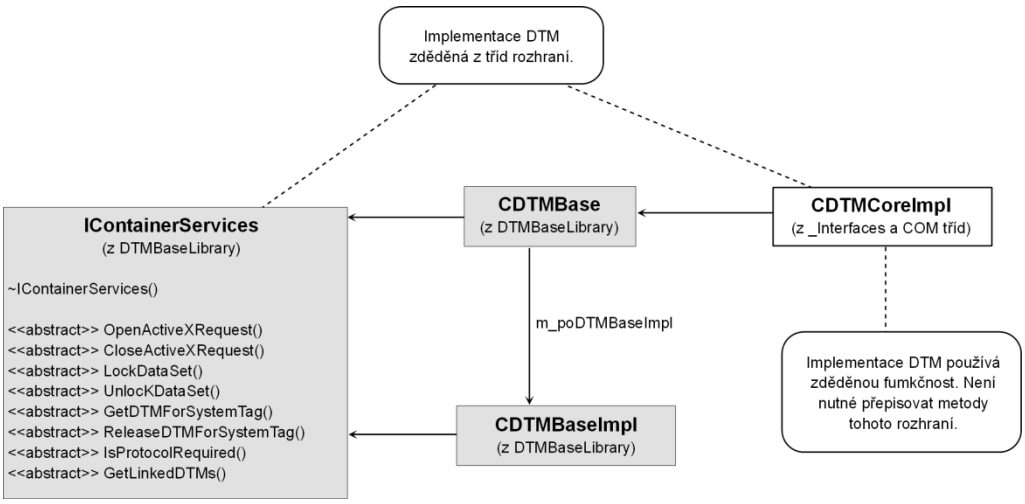
Základní identifikační údaje o přístroji, pro který je DTM určeno, jsou uchovány v XML dokumentu. Mezi jinými jsou to například jméno zařízení, jeho verze a výrobce, podporovaná nebo vyžadovaná sběrnice, na které zařízení funguje apod.




Obrázek 24 - Obsluha typu zařízení

### 7.3.11. Služby rámcové aplikace

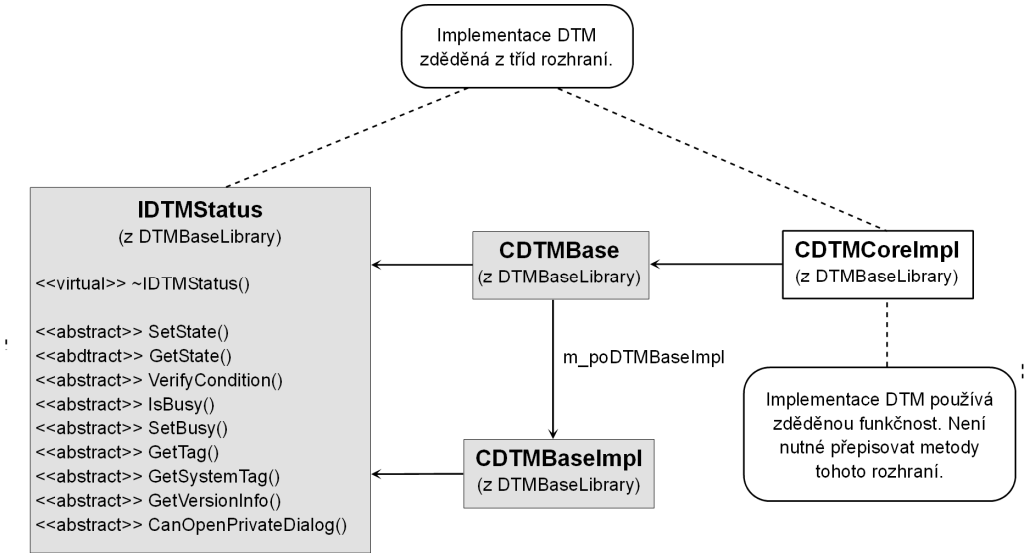
Služby rámcové aplikace, ke kterým může přistupovat DTM, jsou v DTMBaseLibrary funkčně propojeny v rozhraní *IContainerServices*. Jedná se například o práci s ActiveX prvky, manipulaci s datovou sadou, získání jedinečného identifikátoru DTM v systému apod.




Obrázek 25 - Koncepce služeb rámcové aplikace

### 7.3.12. Koncepce rozhraní DTM status

Základní služby spojené s příslušným stavem DTM jsou v DTMBaseLibrary sdruženy do rozhraní *IDTMStatus*. Především se jedná o nastavování jednotlivých stavů, získaní systémového tagu, ověření verze apod.




Obrázek 26 - Koncepce rozhraní DTM status

## 7.4. Grafického rozhraní v dtmManageru

Kromě funkcí pro tvorbu jádra DTM obsahuje dtmManager i funkce pro podporu pro tvorby grafického uživatelského rozhraní, které usnadňuje uživateli návrh aplikace. Tyto funkce jsou ve starší verzi 2.1 umístěny knihovně *DTMBaseLibraryUtilities.dll*, ve verzi novější se pak nacházejí v knihovně *dtmMANAGERSatellite_01.dll*.

Kromě podpory grafických komponent jsou v těchto knihovnách umístěny i funkce pro výměnu dat mezi základním modulem DTM a jeho grafickou nadstavbou, neboť podle filozofie dtmManageru jsou DTM a jeho grafická část rozdělené do dvou nezávislých projektů, jež se následně při běhu DTM v rámcové aplikace vzájemně volají.

### 7.4.1. Tvorba grafického rozhraní

Obecně existuje více možností, jak vytvořit grafické prostředí pro DTM. Je možné klasicky použít grafickou knihovnu MFC (Microsoft Foundation Class Library) nebo WTL (Windows Template Library).

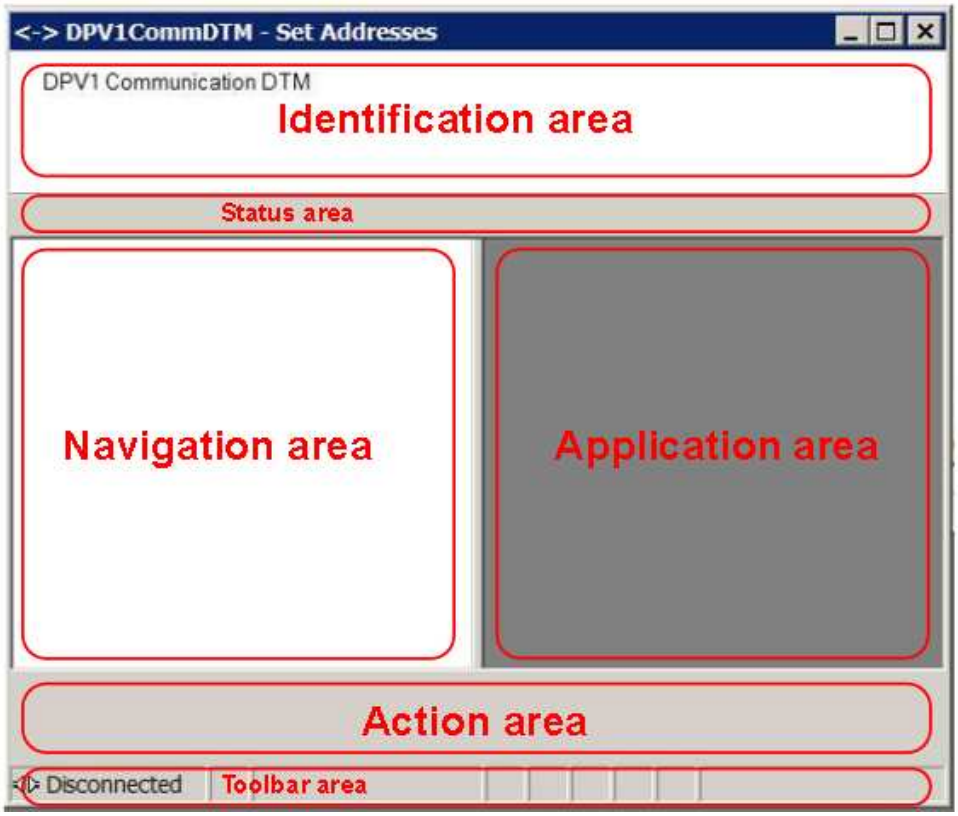
dtmManager však nabízí další možné řešení – na základě WTL nabízí sadu předdefinovaných grafických komponent, které lze v duchu FDT používat skrze strukturovaný XML dokument. Uživatel tedy vytvoří XML strukturu, která v sobě obsahuje popis grafického rozhraní, a funkce dtmManageru se postarají o registraci a vykreslení všech grafických komponent podle představ uživatele.

Použití XML pro tvorbu grafického rozhraní s sebou nese mnohé výhody, ale rovněž i četná úskalí. Výhodou takovéhoto systému je maximální jednoduchost a srozumitelnost. Nevýhodou je, že existuje pouze omezený počet komponent, které uživatel může použít. Rovněž je uživatel svázán danou strukturou, které se musí držet – nemá tedy takovou volnost při návrhu, jakou by měl při návrhu klasickými způsoby.

### 7.4.2. Grafické oblasti

dtmManager definuje pro účely tvorby grafického rozhraní šest různých oblastí, každou se svým specifickým účelem:

- **Identification Area** – oblast slouží k zobrazení základních informací jako je název DTM, výrobce či verze DTM
- **Status Area** – oblast pro speciální funkce DTM
- **Navigation Area** – oblast obsahující stromovou strukturu jednotlivých grafických listů, v kterých se zobrazují informace o zařízení
- **Application Area** – oblast, v které se zobrazují jednotlivé listy
- **Action Aarea** – oblast obsahují základní ovládací prvky pro DTM
- **Toolbar Area** – oblast, ve které se zobrazují ikony stavu zařízení




Obrázek 27 - Rozložení grafických oblastí

Nejdůležitější z těchto uvedených oblastí jsou Navigation Area a Application Area. Právě tyto oblasti lze v dtmManageru vytvořit na základě popisu v XML dokumentu.

V dtmManageru je Navigation Area implementována jako stromová struktura jednotlivých listů příslušných oblasti Application Area. V těchto listech jsou pak dále zobrazovány jednotlivé grafické komponenty a informace o zařízení.

Kořenový element, kterým jsou v XML dokumentu popsány tyto grafické oblasti, je <UILayoutDescription>. Tento element obsahuje další dva elementy <Navigation Area> a <Application Area>, které specifikují oblasti příslušného jména. Element <Navigation Area> specifikuje kořenovou strukturu pomocí elementu <Entry> a <Application Area> specifikuje v elementu <Generic View> jednotlivé listy, které jsou v této oblasti zobrazovány.

dtmManager nabízí v oblasti Application Area dvě možná předdefinovaná prostředí. První se nazývá „SingleColumnLayout“ a je tvořena žebříčkem jednotlivých řádků. Do každého řádku je potom automaticky vložena jedna grafická komponenta definovaná v XML dokumentu. Druhé prostředí se nazývá „DummyLayout“. V podstatě se jedná o prázdný list bez jakéhokoli formátování. Uživatel musí tedy v XML dokumentu přesně definovat pozici, na které se bude daná grafická komponenta zobrazovat.

```
<UILayoutDescription>
    <NavigationArea hide='0'>
        <Entry label='Tutorial DTM' assignedApplicationViewId='0'>
            <Entry label='Parameters DTM' assignedApplicationViewId='1' />
            <Entry label='Manually created dialog' assignedApplicationViewId='2' />
        </Entry>
    </NavigationArea>
    <ApplicationArea>
        <GenericView viewId='1' layoutManager='SingleColumnLayout'>
            <FdtTextControl controlId='1' label='TAG_DESC' tag='tag'>
                <Validator type='stringLength' min='1' max='30' />
            </FdtTextControl>
            <FdtTextControl controlId='2' label='DEVICE_ID' tag='deviceId' />
            <FdtTextControl controlId='3' label='DEVICE_MAN_ID' tag='deviceManufacturerId'>
                <OverwriteSize controlWidth='200' />
            </FdtTextControl>
            <FdtBitEnumControl controlId='4' label='BIT_ENUM_CONTROL' tag='testBitEnum'>
                <Bits>
                    <Bit label='Bit 0' bit='0' />
                    <Bit label='Bit 1' bit='1' />
                    <Bit label='Bit 3' bit='3' />
                </Bits>
                <Validator type='myValidator' />
            </FdtBitEnumControl>
            <FdtComboBoxControl controlId='5' label='DUMMY' tag='dummy'>
                <OverwriteSize controlWidth='200' />
                <ChoiceCollection>
                    <Choice caption='Entry 1' tag='entry1' />
                    <Choice caption='Entry 2' tag='entry2' />
                    <Choice caption='Entry 3' tag='entry3' />
                </ChoiceCollection>
            </FdtComboBoxControl>
            <FdtImageControl controlId='6' resourceId='102' width='200' height='200' />
        </GenericView>
        <GenericView viewId='0' layoutManager='SingleColumnLayout' />
    </ApplicationArea>
</UILayoutDescription>
```

Obrázek 28 - Příklad XML dokumentu popisujícího grafické prostředí

### 7.4.3. Grafické komponenty

dtmManager ve verzi 2.3 nabízí uživateli pět grafických komponent, které může využít při tvorbě svého grafického prostředí:

- **FdtTextControl** – Grafická komponenta skládající se z buňky, v které se zobrazují informace, a popisku.
- **FdtComboBoxControl** – Grafická komponenta, jež v dtmManageru představuje ComboBox.
- **FdtBitEnumControl** – Grafická komponenta obsahující grafické prvky typu CheckBox a jejich popisky.
- **FdtButtonControl** – Grafická komponenta představující tlačítko.
- **FdtImageControl** – Grafická komponenta představující obrázek.

Těchto pět komponent téměř postačuje k vytvoření grafického rozhraní podle přání uživatele. Chybí zde ovšem poměrně důležitá komponenta, které by představovala obyčejný text. Nepřítomnost této komponenty značně znepříjemňuje grafický návrh. Tento nedostatek by měl však již být v dalších verzích dtmManageru odstraněn přidáním dvou grafických komponent **FdtCaptionControl** a **FdtStaticTextControl**.

### 7.4.4. Validace vstupních dat

Pro validaci zadávaných vstupních dat lze v dtmManageru využít zabudovaného validátoru. Ten kontroluje zapsaná data a vyhodnocuje, zda odpovídají předpokládanému formátu. Pokud ne, uživatel je na to upozorněn.

V současné době dokáže validátor kontrolovat, zda je zadaná hodnota v povoleném číselném rozsahu nebo zda obsahuje povolený počet znaků.

# 8. DTM modul CommDTM

## 8.1. Úvod

Jedním z cílů této diplomové práce bylo vytvořit komunikační driver CommDTM, který by umožnil komunikovat rámcové aplikaci FDT a modulům DTM připojených do této aplikace s fyzickými provozními přístroji připojenými ke komunikační sběrnici PROFIBUS PA. Rámcová aplikace umístěná na počítači s operačním systémem Microsoft Windows by se na síť PROFIBUS PA měla připojovat přes komunikační kartu CP-5611 společnosti Siemens, která je umístěna ve slotu PCI.

K vytvoření tohoto modulu byl použit vývojový nástroj pro DTM dtmManager ve verzi 2.1.

## 8.2. Architektura CommDTM

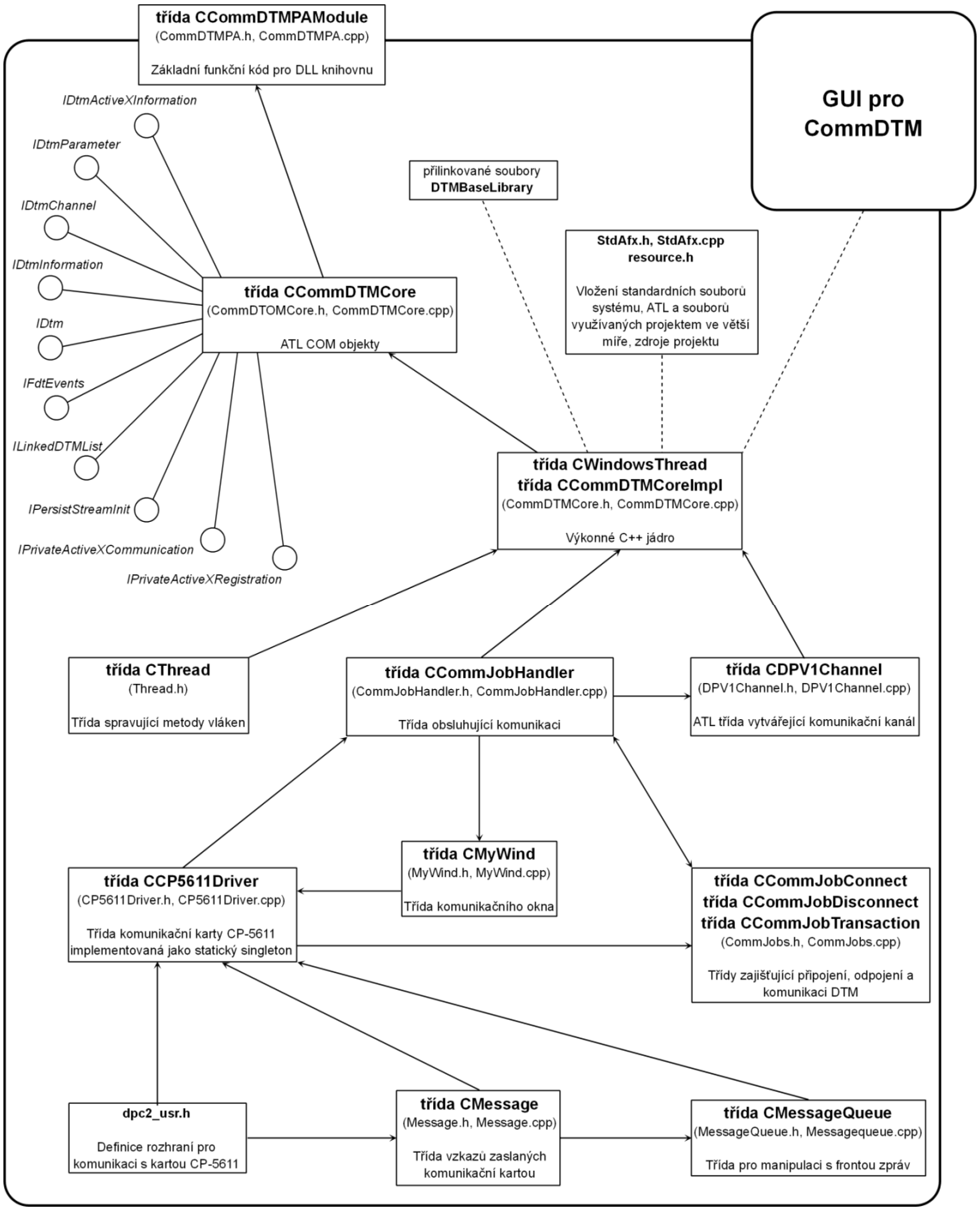
CommDTM je podle principů standardu FDT vytvořen jako dynamická knihovna (dll) v programovacím rozhraní Microsoft Visual C++ s podporou ATL (Active Template Library), COM (Component Object Model) a knihoven DTMBaseLibrary společnosti M&M Software GmbH.

Základní struktura modulu CommDTM vychází ze struktury dynamických knihoven a z principů DTMBaseLibrary – projekt se tedy rozděluje do tří částí. V první části jsou obsaženy metody a deklarace pro ATL COM objekty, pomocí kterých se bude DTM připojovat do rámcové aplikace. V části druhé je obsaženo výkonné jádro CommDTM naprogramované v čistém C++ kódu. Třetí část pak tvoří samostatné grafické prostředí pro tento modul.

Základem každého modulu DTM jsou standardní metody a deklarace příslušné každé dynamické knihovně. V našem případě se jedná o základní třídu CCommDTMPAModule, která je obsažena v souborech *CommDTMPA.h* a *CommDTMPA.cpp*. Jejím návrhem se však nemusí uživatel zabývat, neboť je generována automaticky při tvorbě nového projektu vývojovým prostředím.

ATL COM část projektu je obsažena ve třídě *CCommDTMCore*, jejíž deklarace a metody jsou v souborech *CommDTMCore.h* a *CommDTMCore.cpp*.

Základ výkonného jádra modulu CommDTM tvoří třída CCommDTMCoreImpl, v které jsou uvedeny metody spravující chod DTM. Na tuto třídu jsou navázány třídy *CCommJobHandler*, *CCommJobConnect*, *CCommJobDisconnect*, *CCommJob-Transaction* a *CDPV1Channel*. Třída *CCommJobHandler* se stará o správu jednotlivých komunikačních úkolů, jako je například připojování, odpojování nebo




Obrázek 29 - Architektura CommDTM

provádění transakcí. Tyto jednotlivé úkoly jsou obsaženy v příslušných třídách *CCommJobConnect*, *CCommJobDisconnect* a *CCommJobTransaction*. Ve třídě pojmenované jako *CDPV1Channel* metody pro tvorbu jednotlivých komunikačních kanálů k daným zařízením. Všechny tyto třídy tvoří jádro CommDTM a měly by být obsaženy v každém CommDTM, které je postaveno na principech DTMBaseLibrary.

Kromě těchto tříd zajišťujících běh DTM obsahuje CommDTM ještě třídy, které zajišťují obsluhu karty CP-5611 a komunikaci s ní. Konkrétně se jedná o třídy *CThread*, *CWindowsThread*, *CMyWind*, které spravují komunikační okno pro příjem zpráv od komunikační karty, třídu *CCP5611Driver*, která v sobě obsahuje veškeré metody pro obsluhu a komunikaci s kartou a třídy *CMessage* a *CMessageQueue*, jež představují frontu příchozích zpráv od komunikační karty.

## 8.3. ATL COM část CommDTM

### 8.3.1. Třída CCommDTMCore

Třída *CCommDTMCore* je vytvořena jako ATL COM a představuje spojovací článek mezi DTM a rámcovou aplikací. V této třídě je implementováno volání jednotlivých standardních rozhraní FDT, díky kterým dochází k propojení DTM a rámcové aplikace. Pro funkčnost DTM jako takového byla implementována tato rozhraní (popisy jednotlivých rozhraní naleznete v kapitole 5.3.1):

- *IDtm*
- *IDtmParameter*
- *IDtmInformation*
- *IDtmActiveXInformation*
- *IFdtEvents*
- *IDtmChannel* – Rozhraní potřebné k tomu, aby DTM mohlo fungovat jako CommDTM.

Kromě těchto standardních rozhraní je třeba implementovat ještě další rozhraní, která se nachází v dtmManageru v knihovně *FDT100.dll*:

- *ILinkedDTMList* – Rozhraní nabízené dtmManagerem pro kontrolu připojených DTM (nutné pro CommDTM).
- *IPrivateActiveXInformation* – Rozhraní pro kontrolu ActiveX prvků.
- *IPrivateActiveXRegistration* – Rozhraní pro kontrolu ActiveX prvků.

Jako poslední je ještě implementováno standardní COM rozhraní pojmenované *IPersistStreamInit*, které v DTM slouží pro správu perzistentních dat.

Ve třídě CCommDTMCore dochází kromě implementace rozhraní ještě k nastavení identifikátoru pro systémové registry. Podle těchto identifikátorů potom dokáže rámcová aplikace identifikovat, že daná dynamická knihovna je CommDTM. Nastavení kategorie dynamické knihovny v systémových registrech vypadá pro komunikační DTM takto:

```
BEGIN_CATEGORY_MAP(CCommDTMCore)
    IMPLEMENTED_CATEGORY(CATID_FDT_DTM)
    IMPLEMENTED_CATEGORY(CATID_FDT_COMMUNICATION)
END_CATEGORY_MAP()
```

## 8.4. Výkonné jádro CommDTM

### 8.4.1. Třída CCommDTMCoreImpl

Třída *CCommDTMCoreImpl* představuje základ výkonného jádra modulu CommDTM neboť obsahuje metody callbacků, které zajišťují chod DTM. V tomto DTM je implementováno osm callbackových funkcí, z nichž důležité jsou především tyto callbacky:

- **Callback GetDTMInformation**
  Callback GetDTMInformation v sobě obsahuje XML dokument založený na schématu DTMInformationSchema. V tomto dokumentu jsou popsány základní parametry DTM, jako je například jeho jméno, verze, jméno výrobce, podporovaná sběrnice apod. Tento callback je volán při inicializaci CommDTM v rámcové aplikaci.
- **Callback VerifyDeviceType**
  Tento callback slouží k ověření, zda je toto DTM opravdu určené pro dané zařízení. Pokud ne, uživatel je upozorněn, že se snaží přiřadit DTM k zařízení, které není tímto DTM podporováno.
- **Callback Initialize**
  V tomto callbacku dochází k inicializaci DTM a jeho registraci v rámci FDT kontejneru. V XML dokumentu podle schématu DTMFunctions-Schema jsou zde registrovány funkce, které CommDTM nabízí (v

tomto případě se jedná o funkci, kterou se inicializuje grafické rozhraní pro nastavování adres připojených DTM). Protože se jedná o komunikační DTM, musí se zde rovněž inicializovat instance komunikačního kanálu, který je nezbytný pro komunikaci s kartou.

- **Callback PrepareCommunication**
  Callback PrepareCommunication je volán, pokud se snažíme fyzicky připojit do sítě (tedy měníme stav z offline na online). Dochází zde k vytvoření okna, které slouží k zachytávání zpráv jdoucích od komunikační karty, a následně i k aktivaci samotné karty.
- **Callback PrepareToReleaseCommunication**
  Tento callback slouží k ukončení práce DTM a jeho přechodu ze stavu online na offline, zrušení všech probíhajících komunikačních úkolů a vypnutí komunikační karty.

### 8.4.2. Třída CCommJobHandler

Třída *CCommJobHandler* představuje jakéhosi správce jednotlivých komunikačních úkolů, které při komunikaci probíhají. Pokud CommDTM dostane od připojeného DTM žádost na připojení, odpojení nebo vytvoření transakce, potom je volána jedna z funkcí CommJobHadleru:

- **Funkce CreateConnectJob()** – Tato funkce se snaží připojit DTM k danému zařízení.
- **Funkce CreateDisconnectJob()** – Tato funkce se snaží odpojit DTM od zařízení.
- **Funkce CreateTransactionJob()** – Tato funkce vytváří komunikační kanál pro zápis nebo čtení ze zařízení.

### 8.4.3. Třídy typu CommJobs

Třída *CCommJobHandler* popsaná v předchozí kapitole slouží pouze jako správce jednotlivých komunikačních úkolů. K samotnému zpracování těchto úkolů dochází ve třídách typu CommJobs. Existují tři třídy typu CommJobs, každá odpovídající příslušnému druhu úkolu, který zpracovává:

- **Třída CCommJobConnect**
  V metodách třídy *CCommJobConnect* dochází k inicializaci komunikační karty a vytvoření specifického spojení mezi CommDTM a kartou, které může být dále využito pro komunikaci.

- **Třída CCommJobDisconnect**
  V metodách třídy *CCommJobDisconnect* dochází k odpojování jednotlivých komunikačních spojení od karty.
- **Třída CCommJobTransaction**
  Metody třídy *CCommJobTransaction* slouží k zápisu dát do zařízení nebo naopak ke čtení dat ze zařízení.

### 8.4.4. Třída CDPV1Channel

Metoda *Initialize()* třídy *CDPV1Channel* vytváří objekt komunikačního kanálu, díky kterému je možné komunikovat se zařízeními připojených na sběrnici komunikující podle protokolu PROFIBUS DPV1.

## 8.5. Obsluha komunikační karty

### 8.5.1. Komunikační karta CP-5611

Za účelem komunikace mezi rámcovou aplikací umístěnou na PC a zařízeními připojenými na sběrnici PROFIBUS PA byla zvolena komunikační karta CP-5611 společnosti Siemens.

Aby bylo možné kartu CP-5611 použít, je třeba k projektu CommDTM připojit několik souborů příslušných k této kartě. V první řadě se jedná o hlavičkový soubor *dpc2_usr.h*, který obsahuje deklarace proměnných a metod, které může uživatel použít k ovládání této karty. Souborem potřebným pro kompilaci projektu využívajícím funkce komunikační karty CP-5611 je statická knihovna *dpc2lib.lib*. Pro běh CommDTM používající ke komunikaci tuto komunikační kartu je nezbytné použít dvě dynamické knihovny – knihovnu *DPC2LIB.dll* a knihovnu *S7ONLINX.dll*.

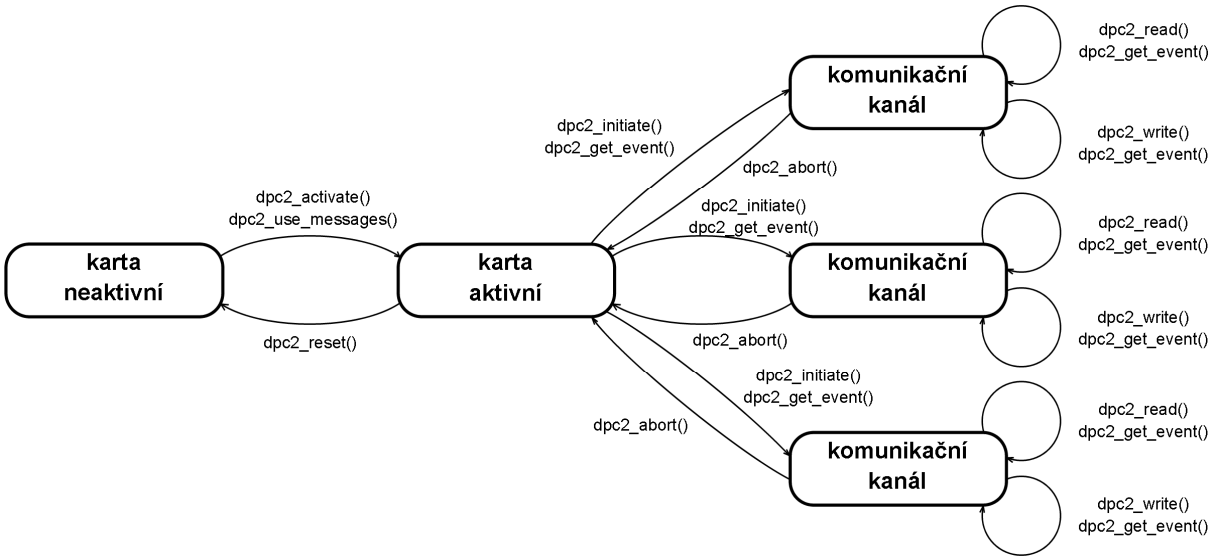
Aby bylo možné komunikovat s kartou CP-6511 a potažmo tedy i se zařízeními, které jsou přes kartu připojeny, je třeba nejdříve tuto kartu aktivovat. Toho uživatel docílí voláním funkce *dpc2_activate()*. Jako vstupní parametry musí uživatel zadat adresu karty nastavenou PG/PC Interface, typ použité komunikace (synchronní nebo asynchronní) a ukazatel na strukturu chybových zpráv. Návratová hodnota této aktivační funkce je identifikátor karty, který je dále využíván pro směrování dalších funkcí na tuto kartu.

Chceme-li komunikovat s kartou asynchronně (což chceme, neboť kartu využíváme jako master třídy 2), musíme funkcí *dpc2_use_messages()* určit, že chceme zasílat zprávy od karty. V operačním systému Microsoft Windows dokáží přijímat zprávy pouze okna. V CommDTM však žádné takové okno nemáme a proto se musí v samostatném vlákně vytvořit neviditelné okno, které bude sloužit pro příjem zpráv zaslaných kartou a z kterého budeme tyto zprávy číst.

Pro otevření komunikačního kanálu k příslušnému zařízení slouží funkce *dpc2_initiate()*. Informace potřebné pro připojení jsou uloženy ve struktuře *dpc2_initiate_rb* a tato struktura je inicializační funkcí zaslána kartě, která podle dat uložených v této struktuře vytvoří příslušné spojení. Jakmile karta dokončí inicializaci spojení, zašle o tom uživateli zprávu, kterou je možné vybrat pomocí funkce *dpc2_get_event()*.

Je-li inicializován komunikační kanál, lze již používat klasické komunikační funkce jako je čtení (*dpc2_read()*), zápis (*dpc2_write()*) nebo výměna dat (*dpc2_data_transport()*). Pro jednotlivé operace je opět třeba vyplnit informační strukturu příslušnou dané operaci (*dpc2_read_rb*, *dpc2_write_rb* nebo *dpc2_data_transport_rb*) a odezvu karty vyzvednou pomocí funkce *dpc2_get_event().*

Ke zrušení příslušného komunikačního kanálu slouží funkce *dpc2_abort()* a její struktura *dpc2_abort_rb*. Chceme-li celkově ukončit práci s kartou použijeme funkci *dpc2_reset()*. Tím se tato karta stane neaktivní.




Obrázek 30 - Použití komunikační karty CP-5611

### 8.5.2. Okno zpráv

Aby bylo možné přijímat zprávy, které zašle uživateli komunikační karta, musíme vytvořit komunikační okno, které bude tyto zprávy zachytávat. Toto okno implementované jako objekt ATL ve třídě *CMyWind* dokáže zpracovávat tři typy zpráv.

Prvním typem zprávy je reakce na událost zavření okna vyvolaná příchozí zprávou s identifikátorem WM_CLOSE. Na tuto akci okno reaguje vysláním požadavku na ukončení všech probíhajících spojení s kartou a její deaktivaci. Po zavření okna následuje jeho zrušení, které je oknu oznámeno zprávou s identifikátorem WM_DESTROY. Z hlediska uživatele je nejdůležitější zprávou zpráva příchozí od komunikační karty, která je označena identifikátorem WM_DPC2MESSAGE. Reakcí na tuto událost je volání funkce, která příchozí zprávu vloží do fronty zpráv.

Aby bylo možné použít okno zpráv pro komunikaci, musí být implementováno v samostatném vlákně, aby nebránilo v běhu výkonnému jádru CommDTM. Toto vlákno je tvořeno dvěma třídami – *CThread* a *CWindowsThread*.

### 8.5.3. Instance karty CP-5611

Instance komunikační karty CP-5611 je implementována ve třídě *CCP5611Driver* a je vytvořena jako statický singleton. To znamená, že v celém projektu existuje pouze jedna jediná instance této třídy, která reprezentuje komunikační kartu.

Instance karty umožňuje připojit až 126 zařízení, což koresponduje se skutečností, že PROFIBUS umožňuje adresovat rovněž až 126 různých zařízení. Přehled o připojených zařízeních je udržován ve statickém poli zařízení, kde jsou pro každé připojené zařízení udržovány informace o jeho aktivitě a rovněž jsou zde uloženy reference na komunikační spojení, které slouží k jednoznačnému určení komunikačního kanálu mezi DTM, kartou a zařízením. Podle této reference lze následně určit, s kterým zařízením je právě komunikováno.

Metody definované v této třídy slouží k ovládání a komunikaci s komunikační kartou. Důležité jsou především tyto funkce:

- **Funkce createCP5611DriverWindow()**
  V této funkci dochází k vytvoření okna zpráv, které je nezbytné pro příjem zpráv od komunikační karty. Bez tohoto okna není možné komunikovat s kartou a tedy ani s připojenými provozními přístroji.
- **Funkce OnMessage()**
  Tato funkce je volána, pokud okno zpráv zaregistruje událost s identifikátorem WM_DPC2MESSAGE. Funkce vytvoří zprávu a tuto zprávu umístí do fronty zpráv pro použití dalšími funkcemi třídy CCP5611Driver().
- **Funkce ActivateCP5611()**
  Voláním této funkce dochází k aktivaci komunikační karty a její nastavení pro asynchronní komunikaci, neboť chceme, aby PC stanice (a potažmo i komunikační karta) pracovala na sběrnici jako master třídy 2.
- **Funkce InitiateCP5611()**
  Funkce *InitiateCP5611()* slouží k inicializaci komunikačního spojení s připojeným zařízením. Pokud je inicializace úspěšná, jsou parametry vytvořeného spojení zaneseny do pole připojených zařízení. Tyto parametry jsou poté použity při komunikaci s daným zařízením.
- **Funkce AbortCP5611()**
  Tato funkce ukončí existující spojení se zařízením a vyřadí zařízení z pole zařízení. Pro další komunikaci se zařízením je nutné spojení nejdříve znovu inicializovat.
- **Funkce ResetCP5611()**
  Pomocí této funkce dochází k celkové deaktivaci karty. Pokud chceme pomocí karty opět komunikovat s nějakým zařízením, musíme ji opět aktivovat pomocí funkce *ActivateCP5611().*
- **Funkce ReadCP5611()**
  Tato funkce je určena pro čtení dat ze zařízení prostřednictvím komunikační karty.
- **Funkce WriteCP5611()**
  Pomocí této funkce lze zapisovat na zvolenou pozici v daném zařízení.
- **Funkce ErrorDisplay()**
  Tato funkce je volána, pokud některá z předchozích funkcí neproběhne korektně a vrátí chybový kód. Funkce identifikuje chybový kód a zobrazí hlášení o chybě.

### 8.5.4. Fronta zpráv

Když okno zpráv dostane zprávu od komunikační karty, je volána funkce *OnMessage()* z třídy *CCP5611Driver*. V této funkci je vytvořena instance zprávy, která je tvořena třídou *CMessage*, a do této zprávy jsou uloženy informace přijaté od karty. Dále je tato zpráva vložena do fronty zpráv.

Fronta zpráv je tvořena třídou *CMessageQueue*. Tato třída vytváří z jednotlivých zpráv spojový seznam. Jednotlivé zprávy uložené ve frontě jsou potom přístupné dalším funkcím spravujícím běh karty CP-5611.

Pro údržbu a správu fronty zpráv byly implementovány tyto funkce:

- **Funkce addMessage()**
  Funkce addMessage() slouží ke vložení nové zprávy do fronty zpráv.
- **Funkce remMessage()**
  Tato funkce slouží k odstranění vybrané zprávy z fronty zpráv.
- **Funkce findMessage()**
  Pomocí této funkce je ve frontě zpráv vyhledána zpráva s příslušným identifikátorem.

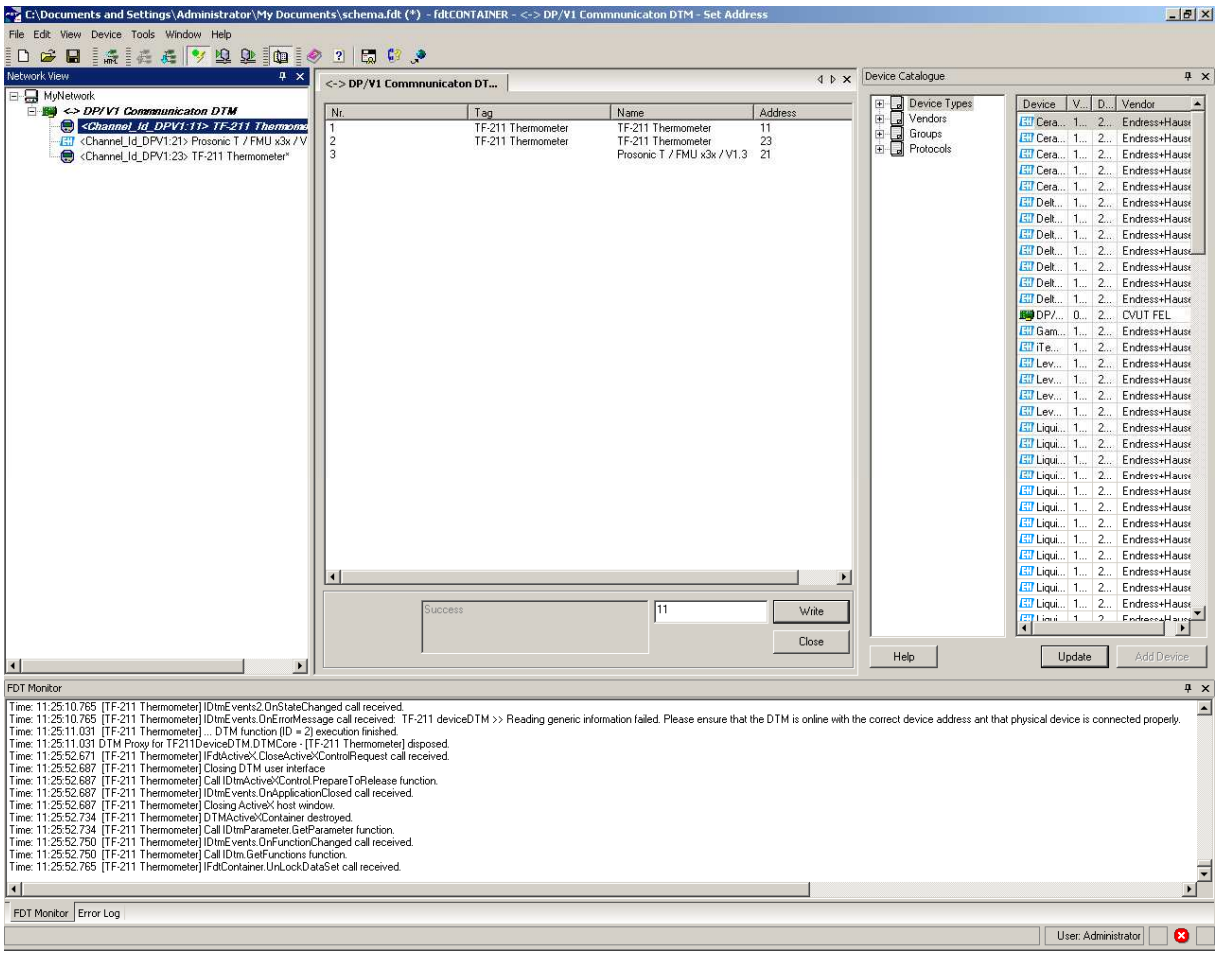
Zprávy s identifikátorem od 0 do 126 mají význačný charakter a slouží jako systémové zprávy vztahující se k zařízení připojenému v řídícím systému na adresu shodnou s tímto identifikátorem. Zprávy s identifikátorem vyšším jsou zprávy vztahující se k jednotlivým úkolům, jako je například čtení nebo zápis.

## 8.6. Grafické rozhraní pro CommDTM

CommDTM nabízí uživateli nastavení adres připojených na sběrnici pomocí grafického uživatelského rozhraní. Pokud nejsou tyto adresy nastaveny na odpovídající hodnoty, není možné s daným zařízením korektně komunikovat.

Protože grafické třídy v dtmManageru verze 2.1 nejsou ještě zcela funkční, bylo grafického rozhraní implementováno ve Visual Basicu a jako jeho základ byl použit tutoriálový příklad z dtmManageru.

Grafické rozhraní využívané CommDTM je umístěno v dynamické knihovně *DTMSampleGUI.ocx*. Kromě této knihovně je ještě pro běh grafického prostředí statická knihovna *DTMSampleGUI.lib*.




Obrázek 31 - Rámcová aplikace s grafickým rozhraním pro CommDTM

Grafické rozhraní spustíme tak, že z výběru funkcí, které nám CommDTM nabízí, zvolíme z nabídky *Additional Functions* položku *Set Address*. Tím aktivujeme okno grafického rozhraní. V hlavním okně je vypsán seznam jednotlivých připojených DTM a jejich adresa, ke které jsou připojeny. Adresa musí odpovídat adrese, na které je zařízení připojeno v síti PROFIBUS. Není-li adresa správná, nastavíme její správnou hodnotu a pomocí tlačítka „Write“ ji zapíšeme do paměti CommDTM. Jsou-li u všech zařízení, se kterými chceme komunikovat, nastaveny adresy, můžeme okno zavřít. CommDTM je připraveno ke komunikaci.

# 9. DTM modul DeviceDTM

## 9.1. Úvod

Posledním z dílčích cílů diplomové práce bylo vytvořit driver pro zařízení na měření teploty, které pracuje jako PROFIBUS PA slave ve verzi 3.0. Tento driver by měl fungovat jako DeviceDTM založený na standardu FDT ve verzi 1.2.1. Modul DeviceDTM se k vybranému provoznímu přístroji prostřednictvím komunikačního modulu CommDTM a komunikační karty CP-5611.

Jako provozní přístroj, ke kterému má být ovladač vytvořen, bylo vybráno teplotní čidlo TF-211 společnosti ABB Automation GmbH. Pro vytvoření tohoto ovladače byly použity knihovny a funkce dtmManageru verze 2.3

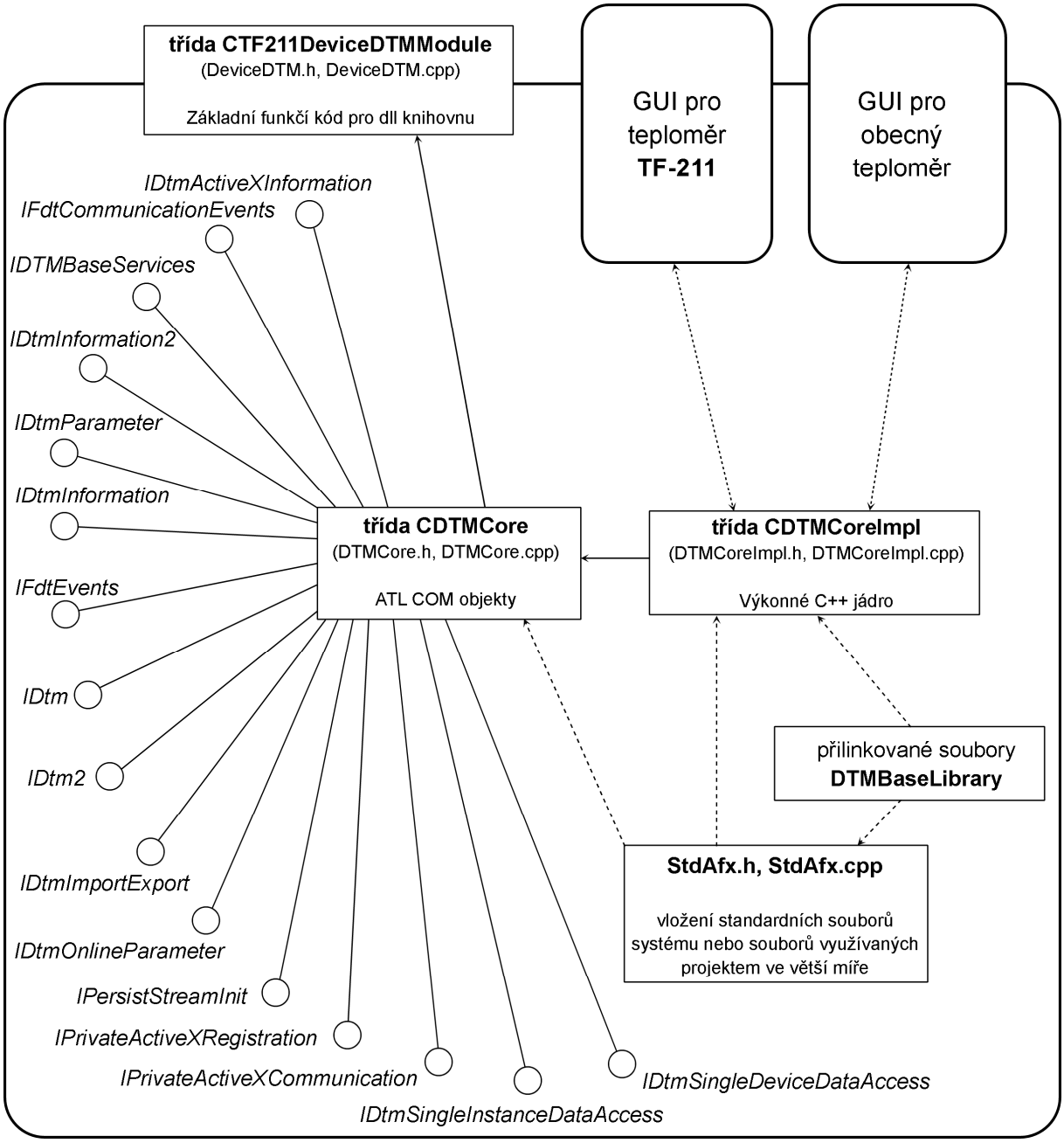
## 9.2. Architektura DeviceDTM

Architektura DeviceDTM je principielně podobná jako u CommDTM. DeviceDTM pracuje jako dynamická knihovna vytvořená v Microsoft Visual C++ s podporou ATL, COM a knihoven DTMBaseLibrary.

Struktura modulu DeviceDTM je v základu stejná jako u CommDTM. První část projektu je tvořena ATL COM třídou, které se stará o komunikaci s nadřazenou rámcovou aplikací. Druhou část projektu tvoří výkonné jádro DeviceDTM. Třetí část je tvořena grafickým rozhraním, které je specifické pro teploměr TF-211. Poslední částí projektu je grafické rozhraní pro jakýkoli teploměr, který pracuje jako PROFIBUS PA slave profilu 3.0 nebo 3.01.

Základ DeviceDTM je tvořen třídou *CTF211DeviceDTMModule*, která v sobě obsahuje metody a proměnná nutné pro běh projektu jako dynamické knihovny. Tato třída je obsažena v souborech *TF211DeviceDTM.cpp* a *TF211DeviceDTM.h.* ATL COM část projektu můžeme nalézt ve třídě *CDTMCore*, která obsahuje všechna rozhraní potřebná pro propojení DeviceDTM a rámcovou aplikací. Výkonné jádro DTM je opět podle principu DTMBaseLibrary umístěno ve třídě *CDTMCoreImpl*.

Na první pohled je struktura DeviceDTM výrazně jednoduší než u CommDTM. DeviceDTM sice neobsahuje tolik specializovaných tříd jako CommDTM, ale o to rozsáhlejší a masivnější je jeho jádro.




Obrázek 32 - Architektura DeviceDTM

## 9.3. ATL COM část DeviceDTM

Třída *CDTMCore* tvoří podobně jako třída CCommDTMCore spojení mezi DTM a rámcovou aplikací. Třída v sobě obsahuje volání jednotlivých rozhraní FDT, které slouží pro správu DTM v rámcové aplikaci a obsluze jednotlivých funkcí DTM, jež jsou na FDT navázány. Rozhraní, která byla implementovaná v DeviceDTM, můžeme nalézt v obrázku 32. Tato rozhraní jsou popsána v kapitole 5.3.

Nastavení kategorie dynamické knihovny, kterou je DeviceDTM tvořen, se zde provádí podobně jako u CommDTM. Po registraci dynamické knihovny do systémových registrů poté dokáže rámcová aplikace podle těchto identifikátorů mezi dalšími dynamickými knihovnami nalézt DTM. Registrace kategorie dynamické knihovny vypadá pro DeviceDTM takto:

```
BEGIN_CATEGORY_MAP(CDTMCore)
    IMPLEMENTED_CATEGORY(CATID_FDT_DTM)
    IMPLEMENTED_CATEGORY(CATID_FDT_DEVICE)
END_CATEGORY_MAP()
```

## 9.4. Výkonné jádro DeviceDTM

Ve vytvořeném CommDTM tvoří výkonné jádro více rozdílných tříd umístěných v různých souborech. V DeviceDTM je výkonné jádro tvořeno pouze jednou jedinou třídou *CDTMCoreImpl*, v které jsou implementovány všechny funkce, jež jsou pro běh tohoto DTM třeba.

Ve výkonném jádře DeviceDTM je množství funkcí callbacků. Tyto funkce slouží především jako reakce na události vyvolané rámcovou aplikací, jako je například, inicializace, komunikace apod. Některé callbacky jsou pro chod DTM nezbytné, jiné pouze nabízejí další rozšiřující funkce, které nejsou pro základní běh třeba.

Protože DeviceDTM je založeno na novější verzi dtmManageru než CommDTM, může se tělo některých callbacků odlišovat, i když princip těchto callbacků je zcela totožný jako u CommDTM. Popis těchto callbacků je uveden v kapitole 8.4.1 a proto zde nebude uveden.

V DeviceDTM je možné nalézt tyto callbacky:

- **Callback GetDTMInformation**
- **Callback VerifyDeviceType**
- **Callback Initialize**
- **Callback PrepareCommunication**
- **Callback PrepareToReleaseCommunication**
- **Callback SetParameterDocument**
  Tento callback je volán, když chce rámcová aplikace aktualizovat XML dokument, v kterém jsou obsaženy informace o tomto DTM

- **Callback GetParameterDocument**
  Tento callback je volán, chce-li DTM upravit dokument se svými parametry. Ze strany DTM je možné upravovat pouze tag DTM a adresu sběrnice, na kterou je toto DTM připojeno.
- **Callback InvokeFuction**
  Callback InvokeFuction se používá ke spuštění specializovaných funkcí DTM. V tomto DeviceDTM měl původně sloužit ke spouštění aplikace, která
- **Callback InitializeLogging**
  Tento callback složí uživateli k nastavení výpisu událostí, ke kterým dochází v DTM a rámcové aplikaci. V našem případě je výpis ukládán do souboru „C:\TF211DTM.log“ a jsou ukládány informace úrovně LOG_LEVEL_TRACE. Označení události, která je určena k výpisu, vypadá například takto:

```
CGlobal::GetLogger().LogMessage(LOG_LEVEL_TRACE,
m_oMyPersistentTag.GetString(),
SOURCE_CONTEXT(L"CDTMCoreImpl::Callback_InvokeFunction"),
L"Function call", L"Function2 called...\n");
```

- **Callback SetLanguage**
  Pomocí tohoto callbacku je možné nastavit, v jakém jazyce bude DTM s uživatel komunikovat. Podmínkou je, že texty musí být připravené pro použití ve více jazycích. Změna nastavení jazyka pro DTM se nevztahuje na změnu jazyka v uživatelském rozhraní – uživatelské rozhraní je samostatný projekt.
- **Callback GetItem**
  Callback GetItem patří mezi nejdůležitější callbacky obsažené v DeviceDTM. Pokud uživatel dá v grafickém rozhraní pokyn k čtení dat ze zařízení, je volán tento callback. Jeho vstupní hodnotou je identifikátor, který callbacku říká, jaký údaj si přeje uživatel číst. Pokud je čtení za zařízení úspěšné, callback vrací přečtený údaj v podobě řetězce znaků a zasílá ho rámcové aplikace, která ho přepošle aplikace, z které vzešel požadavek pro čtení.
  Samotné čtení ze zařízení probíhá ve funkci *readFromDevice()*, která je z callbacku GetItem volána. Čtení ze zařízení probíhá tímto způsobem:

```
try {
   VerifyCondition(CDTMStateMachine::IDtmOnlineParameter);

   m_oPAUtilities.ObtainOwnership(INFINITE);
   m_oCommunicationProvider.GetResponseBuffer().DispatchMessagesOnWait(true);

   if (!m_oCommunicationProvider.IsConnected()) {
      m_oPAUtilities.OpenDeviceConnection(m_oBusInformation.GetBusAddress(), 0);
   }

   if (tag == _bstr_t(L"header")) {
      oItemValue = m_oPAUtilities.ReadFromDevice(1, 0);
   }
}
catch(...) {
   // unable to read, show a message
   _bstr_t oMessage = "TF-211 deviceDTM >> Reading generic information failed. "
                      "Please ensure that the DTM is online with the correct"
                      "device address ant that physical device is connected "
                      "properly.";

   SignalError(oMessage);
}

m_oPAUtilities.ReleaseOwnership();
```

DTM nejdříve zkontroluje, zda je ve stavu online. Pokud je tato podmínka splněna, převezme správu nad komunikačním kanálem a pokud je DTM připojeno, je se zařízením vyskytujícím se na dané adrese navázáno spojení. Po navázání spojení je možno číst z příslušného slotu a indexu zařízení. Po přečtení dat ze zařízení je ještě třeba uvolnit správu nad komunikačním kanálem, aby byl připraven pro další použití.

- **Callback SetItem**

  Callback SetItem je primárně určen k zápisu dat do zařízení. Je-li vydán pokyn k zápisu, je volán právě tento callback. Vstupními parametry tohoto callbacku jsou identifikátor, podle kterého callback rozlišuje, na jakou adresu má zařízení má zapisovat, a data v podobě řetězce znaků, která přijdou na danou adresu zapsat.

  Tento callback nemusí být nutně využit jen pro přenos dat určených k zápisu do zařízení, ale lze ho rovněž využít pro přenos uživatelských dat, která jsou určena pro správu běhu DTM. V našem případě je

tohoto callbacku využito pro přenos parametru, který slouží k nastavení frekvence cyklického čtení teploty ze zařízení.
Zápis do zařízení probíhá podobně jako čtení:

```
try {
   VerifyCondition(CDTMStateMachine::IDtmOnlineParameter);

   m_oPAUtilities.ObtainOwnership(500);
   m_oCommunicationProvider.GetResponseBuffer().DispatchMessagesOnWait(true);

   if (!m_oCommunicationProvider.IsConnected()) {
      m_oPAUtilities.OpenDeviceConnection(m_oBusInformation.GetBusAddress(), 0);
   }

   if (tag == _bstr_t(L"tag107") && (_bstr_t)oItemValue != L"NO_READ_DATA") {
      m_oPAUtilities.WriteToDevice(slotFB, indexFB+11,_bstr_t(oItemValue));
      bSuccess = true;
   }
}
catch(...) {
   // unable to write, show message
   _bstr_t oMessage = "TF-211 deviceDTM >> Writing the tag failed. Please "
                      "ensure that the DTM is online with the correct device "
                      "address and that the physical device is connected "
                      "properly. The tag also must not be empty....";
   SignalError(oMessage);
}

m_oPAUtilities.ReleaseOwnership();
```

Kromě callbacků obsahuje DTM několik dalších funkcí, které slouží k provádění speciálních akcí. Nejdůležitější z těchto funkcí je funkce, jež slouží k cyklickému čtení dat ze zařízení. Volbu doby, za kterou dojde k obnově dat, volí uživatel v grafickém rozhraní. Tato hodnota je pomocí callbacku SetItem přenesena z grafického rozhraní do jádra DTM a podle své hodnoty dále zpracovávána.

Aby bylo možné cyklicky číst ze zařízení, je třeba spustit tuto operaci v samostatném vláknu, jinak by došlo k odstavení ostatních funkcí jádra a DTM by přestalo reagovat. Pokud uživatel pustí cyklické obnovování, je volána funkce *_ThreadRequest()*, v které dochází ke spuštění samostatného vlákna. V tomto vlákně běží nekonečná smyčka umístěná ve funkci *Thread()*. V této smyčce dochází k opakovanému čtení ze zařízení s frekvencí vybranou uživatelem.

## 9.5. Uživatelské rozhraní pro TF-211

Základní jádro DTM lze obecně využít pro všechna zařízení stejného typu. To, co dělá z DeviceDTM individuální ovladač pro dané zařízení, je až uživatelské rozhraní a funkce vyplývající z potřeb tohoto rozhraní, neboť každé zařízení má jiné nároky a funkce.

Podle principů dtmManageru je uživatelské rozhraní vytvářeno jako samostatný projekt v Microsoft Visual C++. K návrhu je třeba použít knihovny ATL, WTL, COM a rovněž knihovny dtmManageru. V našem případě byl pro návrh grafického prostředí použit dtmManager ve verzi 2.3.

### 9.5.1. Architektura GUI pro TF-211

Grafické rozhraní pro teploměr TF-211 je vytvořené jako samostatná dynamická knihovna *TF211DeviceDTMGUI.dll*. Architektura této knihovny je značně podobná, jako architektura CommDTM a DeviceDTM. Základ projektu je tvořen třídou *CTF211DeviceDTMGUIModule*, jež obsahuje metody nutné pro běh projektu jako dynamické knihovny.

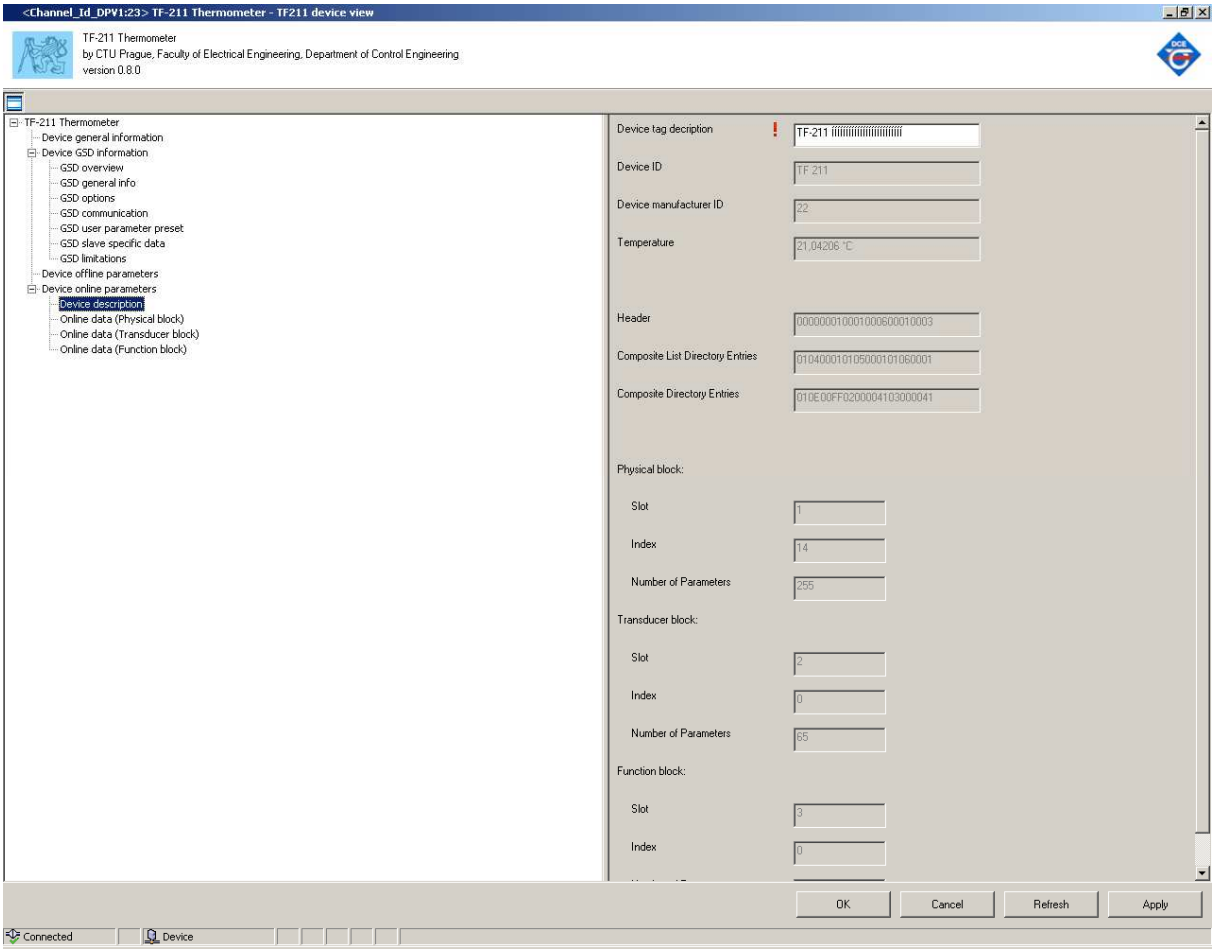
Pro spolupráci s rámcovou aplikací a vytvoření grafického okna, v kterém je umístěno uživatelské rozhraní, slouží ATL COM třída *CDeviceView*. V této třídě jsou umístěna základní rozhraní sloužící ke komunikaci s rámcovou aplikací (*IDtmActiveXControl*), rozhraní pro využití grafických možností dtmManageru (*IUpdateUI* a *CFdtActiveXImpl*) a standardních rozhraní ATL COM, která se používají pro grafické účely. Aby bylo možné v rámcové aplikaci adresovat komunikaci mezi grafickou částí a jádrem DeviceDTM, je třeba nastavit identifikátor ProgId. Tento identifikátor je pro toto uživatelské rozhraní nastaven takto:

```
SetProgId(L"TF211DeviceDTMGUI.DeviceView");
```

Pro samotnou funkčnost grafického rozhraní slouží třída *CEditModel*. V této třídě můžeme naleznout nejen popis grafického prostředí, jenž je vytvořen jako XML dokument, ale i metody, které se starají o reakce na události vzniklé v uživatelském rozhraní.

### 9.5.2. Popis grafického prostředí

Uživatelské prostředí tohoto DeviceDTM plně využívá možností, jež mu nabízí dtmManager, jak bylo popsáno v kapitole 7.4. K základnímu popisu dochází ve funkci *Initialize()*. Zde jsou definovány grafické komponenty, které se zobrazují v jednotlivých grafických oblastech. Pro oblasti Navigation Area a Application Area jsou grafické prvky definované pomocí XML dokumentu. Výsledný návrh uživatelského prostředí je na obrázku 33.




Obrázek 33 - Uživatelské rozhraní DeviceDTM

Uživatelské rozhraní v sobě obsahuje několik grafických listů. Prvním listem je list nazvaný „Device general information“. Vzhled tohoto listu můžeme vidět na obrázku 33. Další položkou je „Device GSD information“. Informace obsažené v těchto listech popisují GSD soubor určený pro teploměr TF-211 a jsou určeny spíše k demonstraci využití komponenty FdtTextControl jako statického textu.

V listu „Device offline parameters“ jsou vytvořeny dva ComboBoxy. Pomocí prvního lze nastavit frekvenci čtení ze zařízení. DeviceDTM nenačítá cyklicky všechny parametry, pouze teplotu. Pokud chceme číst všechny parametry, musíme nastavit frekvenci obnovování údaje na nulu a poté použít tlačítko „Refresh“ umístěné v oblasti Action Area. Protože teploměr TF-211 v sobě nemá definované, v jakých jednotkách bude teplotu zobrazovat (standardně zobrazuje pouze ve stupních Celsia), uživatel si může vybrat, v jakých jednotkách chce měřenou teplotu zobrazit. Převod je prováděn v DTM, nikoli přímo v zařízení.

Posledním položkou je „Device online parameters“. Tato položka obsahuje čtyři grafické listy, které obsahují údaje získané z přístroje:

- **List „Device description“** – Tento list obsahuje základní údaje o zařízení, které byly přečteny. Pro informativní účely zde jsou uvedeny i položky z adresáře objektů a dešifrování položky Composite Directory Entries pro fyzický, převodní a funkční blok.
- **List „Online data (Physical block)“** – V tomto listu jsou uvedeny položky přečtené z fyzického bloku zařízení. Pokud nebyl některý parametr ze zařízení přečten, je zobrazena zpráva NO_READ_DATA.
- **List „Online data (Tranducer Block)“** – Položky zobrazené v tomto listu obsahují data přečtená z převodního bloku zařízení.
- **List „Online data (Function Block)“** – Pro zobrazení parametrů obsažených ve funkčním bloku zařízení slouží tento list.

V oblasti Action Area jsou umístěna čtyři tlačítka. Tlačítko „OK“ vydá pokyn k uložení a zápisu dat do zařízení a ukončí grafické rozhraní. Tlačítko „Cancel“ ukončí grafické rozhraní. Tlačítko „Refresh“ slouží k znovunačtení dat ze zařízení. Pomocí tlačítka „Apply“ je možné uložit změněné položky do zařízení.

V oblasti Toolbar Area se zobrazuje několik základní stavových ikon, které uživatele informují o stavu zařízení, pro které je DeviceDTM určen. Nejdůležitější ikonou je v tomto případě indikace, zda je DTM k zařízení připojeno (connected) nebo nepřipojeno (disconnected). Pokud dochází k cyklickému čtení z přístroje, je to v této oblasti rovněž indikováno.

### 9.5.3. Reakce na události

Jednou z nejdůležitějších funkcí obsažených ve třídě *CEditModel* je funkce *ProcessUIEvent()*. Tato funkce přijímá události, které se staly v uživatelském rozhraní (jako je například změna nějaké grafické komponenty), a vyhodnocuje je. Existuje několik základních událostí, na které lze reagovat:

- **Událost UIE_GETTEXTDATA** – Tato událost se běžně využívá při čtení hodnot ze zařízení a slouží k nastavení hodnoty textových komponent nebo pomocných textových proměnných.
- **Událost UIE_SETTEXTDATA** – Pokud dojde ke změně některého parametru uživatelem, je indikována tato událost.
- **Událost UIE_COMMAND** – Událost COMMAND je volána, pokud je stisknuto nějaké tlačítko umístěné v oblasti Application Area. Podle identifikátoru tohoto tlačítka je možné vykonat akci, pro kterou je tlačítko určeno. V našem případě je reakce na tuto událost využita pro nastavení hodnot ComboBoxů v listu „Device offline Parameters“.
- **Událost UIE_ACTION** – Když uživatel zmáčkne nějaké tlačítko vyskytující se v oblasti Action Area, dojde právě k této události. Podle identifikátorů jednotlivých tlačítek lze potom vykonat příslušnou reakci, jako je ukončení běhu grafického rozhraní, obnova dat, nebo zápis do zařízení.
- **Událost UIE_COMBOBOX_GETTEXT** – K této událost dochází ve chvíli, když uživatel vybere z ComboBoxu nějakou položku. V našem případě je tato událost využita k nastavení pomocných proměnných, které jsou využívány při stanovování frekvence obnovování a převodu jednotek změřené teploty.
- **Událost UIE_COMBOBOX_SETTEXT** – Reakcí na tuto událost je možné získat nastavenou hodnotu ComboBoxu.

Kromě těchto událostí, které využíváme v našem programu, existují ještě dvě další události – UIE_BITENUM_GETVALUE a UIE_BITENUM_SETVALUE. Které slouží k vyhodnocování událostí kolem CheckBoxů. V rámci tohoto projektu jsou však tyto události nevyužité.

### 9.5.4. Čtení ze zařízení

Pokyn ke čtení ze zařízení a obnově dat je vydán, stiskne-li uživatel tlačítko „Refresh“. Reakcí na toto událost je volání funkce *ProcessUIEvent()*, v které je identifikováno, že došlo ke stisku tohoto tlačítka, a je zavolána funkce *Refresh()*.

V těle této funkce dochází k postupnému vysílání požadavků do jádra DeviceDTM, aby došlo ke znovunačtení příslušné položky se zařízení. Funkce, která slouží k vyslání požadavku, je pojmenována jako *GetItemFromDTM()*. Vstupním parametrem této funkce je identifikátor, podle kterého DTM pozná, jakou hodnotu chceme ze zařízení číst. Vrácená návratová hodnota této funkce je hexadecimální řetězec, který byl přečten ze zařízení.

Aby byl uživatel schopen načteným parametrům porozumět, je třeba je z hexadecimálního řetězce převést do čitelného tvaru. Existuje několik druhů převodů v závislosti na tom, jakého je ten který parametr datového typu. Jsou použity převody na tyto základní datové typy:

- **Float** – Hexadecimální řetězec o délce 4 bytů je převeden na reálné číslo v plovoucí řádové čárce podle normy IEEE 754.
- **Unsigned8** – Převod na přirozené číslo v rozsahu 0-255 (1 byte).
- **Unsigned16** – převod na přirozené číslo v rozsahu 0-65535 (2 byty).
- **VisibleString** – Každý byte hexadecimálního řetězce představuje jeden znak zakódovaný v ASCII tabulce. Pro převod do čitelné formy je třeba převést každý byte zprávy
- **OctetString** – Kódování OcstetStringu je stejné jako u VisibleStringu.

Kromě základních datových typů existují ještě datové typy složené. Tyto složené datové typy se skládají ze základních datových typů. Pro jejich převod je nutné je rozdělit a poté jednotlivé části převést do odpovídajícího datového typu.

Po přečtení všech položek ze zařízení je třeba ještě překreslit grafické okno, aby byly změny viditelné. Toho docílíme tímto příkazem:

```
// Refresh all controls!
if (m_poUIController) {
   // This line informs all controls that they are supposed to refresh.
   m_poUIController->OnEditModelChanged(DWORD(-1), DWORD(-1));
}
```

### 9.5.5. Zápis do zařízení

Pro zápis parametrů do zařízení lze použít tlačítko „OK“ nebo „Apply“. Opět je indikována událost stisku tlačítka a vzhledem k vnitřní logice zavolána funkce *OnActionApply().* V této funkci je DTM zaslán požadavek na zapsání položek do zařízení. K vyslání požadavku dochází tímto způsobem, kde je zasílán identifikátor a hodnota, která se má do zařízení zapsat:

```
if (m_oPtrDtmBaseService != NULL) {
   hr = m_oPtrDtmBaseService->SetItem(_bstr_t("Tag"), _variant_t(oTag),
   &result);
}
```

Protože jednotlivé položky, které budeme zapisovat, jsou převedeny do tvaru čitelného člověkem, je třeba je zpětně převést ze základních datových typů do hexadecimálního řetězce odpovídající délky a až tuto upravenou hodnotu předа DTM pro zápis. V případě složených datových typů nejdříve převedeme jednotlivé části, zřetězíme do správného tvaru a teprve poté zašleme DTM.

## 9.6. Uživatelské rozhraní pro obecný teploměr

Uživatelské rozhraní pro obecný teploměr je určeno pro jakýkoli teploměr pracující na PA profilu 3.0 nebo 3.01 a umožňuje číst vnitřní strukturu fyzického, převodního a fyzického bloku a zobrazovat položky odpovídající třídě A. Svojí architekturou vychází toto grafické rozhraní z rozhraní určeného pro teploměr TF-211, jsou zde však vynechány grafické listy, které se týkají přímo teploměru TF-211. Celková funkce i použité metody jinak zůstávají shodné jako u grafického rozhraní pro TF-211.

# 10. Rámcová aplikace a moduly DTM

## 10.1. Volba rámcové aplikace

Pro použití modulů DTM je nezbytné používat nějakou rámcovou aplikaci, které bude tyto moduly spravovat. Ze širokého výběru možných rámcových aplikací jsme nakonec zvolili rámcovou aplikaci fdtContainer 2.1 společnosti M&M Software GmbH. Pro tuto rámcovou aplikace jsme se rozhodli především proto, že je nabízena bezplatně, a rovněž proto, že je od stejné společnosti jako dtmManager, jenž byl použit pro tvorbu modulů DTM, takže je předpokládána vysoká kompatibilita mezi těmito nástroji.

## 10.2. Práce s rámcovou aplikací

### 10.2.1. Device Catalogue

Při prvním spuštění rámcové aplikace je uživatel požádán, zda chce vytvořit katalog DTM. Pokud uživatel zareaguje souhlasně, rámcová aplikace prohledá systémové registry a u každé dynamické knihovny zkontroluje, zda není zaregistrována v příslušné kategorii. Pokud ano, tak ji zařadí do XML dokumentu, který představuje katalog zařízení a který je rámcovou aplikací načítán.

Pokud si uživatel přeje aktualizovat katalog zařízení, může stiskem tlačítka „Update“ spustit nové hledání modulů DTM. Rámcová aplikace neaktualizuje katalog zařízení při svém spuštění, takže tuto operaci je nutné provést, pokud chceme vyhledat nově přidané DTM.

### 10.2.2. Network View

Network View slouží uživateli k vytvoření virtuální reprezentace sítě, kde připojené moduly DTM představují jednotlivá zařízení umístěná na sběrnici. Kořen struktury představuje rámcovou aplikace. K rámcové aplikace je připojen CommDTM, který reprezentuje komunikační kartu. Jednotlivé DeviceDTM se potom připojují ke CommDTM.

### 10.2.3. Monitor View

Monitor View slouží k zobrazovaní informací a volání, které probíhají v rámcové aplikace. Podle těchto údajů uživatel vidí, jaké činnosti v rámcové aplikaci probíhají. Monitor View lze rovněž využít pro zobrazování chybových zpráv pro události, ke kterým došlo v DTM. Pokud bychom pro zobrazení takovéto zprávy použili v DTM například MessageBox, došlo by k zastavení činnosti DTM, dokud by uživatel nepotvrdil přečtená chybové zprávy. Takto může DTM pokračovat dále ve své činnosti.

## 10.3. Práce s moduly DTM

### 10.3.1. Přenositelnost

Moduly DTM je možné použít na libovolném počítači, který je vybaven operačním systémem Microsoft Windows 2000/XP. Aby bylo možné DTM použít, je třeba ho nejdříve zaregistrovat do systému. Protože DTM je dynamická knihovna, použijeme pro registraci příkaz *regsvr32*. Při registraci se do systémových registrů zapíše příslušná kategorie DTM a rámcová aplikace bude potom moci podle této kategorie DTM vyhledat.

Oba vytvořené moduly, CommDTM i DeviceDTM, potřebují ke svému běhu některé další soubory. Bez těchto souborů nebudou tyto soubory fungovat a ani nebude možné je zaregistrovat. Tyto soubory musí být umístěny ve stejném adresáři jako dynamická knihovna DTM.

Aby CommDTM dokázalo spolupracovat s komunikační kartou, potřebuje soubory *DPC2LIB.DLL* a *S7ONLINX.DLL*. Z dtmManageru je dále třeba použít knihovnu *DTMBaseLibraryInterfaces.dll*. Grafické rozhraní pro CommDTM se nachází v souboru *DTMSampleGUI.ocx*.

Pro funkčnost DeviceDTM je třeba zaregistrovat soubor *TF211DeviceDTM.dll*, který představuje jádro DTM, a grafická rozhraní *TF211DeviceDTMGUI.dll* a *GeneralDTMGUI.dll*. Kromě těchto knihoven je pro chod třeba ještě knihovny *dtmManagerSatellite_01.dll*. Kromě těchto souborů obsahuje projekt ještě tři obrázky, na které je odkazováno z vnitřku dynamických knihoven.

Jestliže nejde DTM registrovat nebo nejde inicializovat, lze pomocí programu „depends.exe“, který je obsažen Microsoft Visual Studiu, určit, zda jsou všechny soubory, které daná dynamická knihovna potřebuje ke svému chodu, přítomné (v některých operačních systémech chybí v adresáři System32 dynamické knihovny, které je třeba ručně doplnit).

### 10.3.2. Spuštění DTM

Podaří-li se DTM úspěšně zaregistrovat a inicializovat v rámcové aplikaci, je připraveno k použití. Pomocí tlačítka „Add“ vložíme DTM z katalogu zařízení do virtuální sítě. Jedná-li se o DeviceDTM, nastavíme pomocí uživatelského rozhraní „Set Address“ (v CommDTM) adresu daného zařízení. Ke komunikační kartě se připojíme výběrem funkce „Go online“. Grafické rozhraní lze spustit výběrem z funkce „Additional Fuctions“. Spuštěním grafického rozhraní se zároveň DTM pokusí připojit k zařízení a získat z něj vybrané hodnoty.

# 11. Závěr

Základem diplomové práce bylo seznámit se s principy fungování standardu FDT/DTM a takto získaných znalostí využít při tvorbě komunikačního modulu CommDTM a ovladače zařízení pro měření teploty DeviceDTM. Jelikož přístroj, pro který byl DeviceDTM navrhován, je umístěn na průmyslové sběrnici PROFIBUS PA, bylo rovněž nezbytné seznámit se s principy fungování této sběrnice.

Pro vytvoření ovladačů DTM byl použit vývojový nástroj dtmManager ve verzích 2.1 a 2.3. Tento vývojový nástroj nabízí uživateli velké množství podpůrných funkcí, které slouží ke tvorbě DTM. Bohužel k tomuto nástroji neexistuje kvalitní dokumentace nebo popis API, což programátorovi značně znepříjemňuje návrh, neboť bez této dokumentace nezná všechny možnosti, které mu tento vývojový nástroj nabízí. Tuto nevýhodu jsem se alespoň zčásti eliminoval tím, že jsem vytvořil vlastní popis jednotlivých funkcí dtmManageru verze 2.1.

Komunikační modul CommDTM byl vytvořen s použitím dtmManageru 2.1. Kromě vytvoření základního funkčního jádra modulu v něm byly implementovány funkce, které umožnují tomuto modulu spolupracovat s komunikační kartou CP-5611, přes kterou může tento ovladač komunikovat se zařízeními připojenými na sběrnici.

Při tvorbě ovladačů bylo zjištěno, že verze dtmManageru 2.1 a 2.3 nejsou spolu vzájemně kompatibilní – DTM a jeho grafické rozhraní musí být ve shodné verzi. Proto CommDTM obsahuje pouze jednoduché grafické rozhraní vytvořené ve Visual Basicu, neboť vyšší verze dtmManageru byla získána nedlouho před odevzdáním diplomové práce. Přesto je však CommDTM plně funkční.

V DeviceDTM měl být implementován profil 3.0 pro zařízení PROFIBUS PA na měření teploty. Připojené teplotní čidlo TF-211 je však koncipováno na profilu 2.0. Proto byly vytvořena dvě grafická rozhraní. První se specializuje přímo na teploměr TF-211. Protože však byl k dispozici pouze popis pro profil 3.0, nedokáže DTM v zařízení nalézt některé parametry, popřípadě mají tyto parametry jiný datový typ. Druhé grafické rozhraní je již schopné plně pracovat s jakýmkoli zařízením pro měření teploty třídy A založeným na profilu 3.0 nebo 3.01. Jádro DeviceDTM je pro obě různá rozhraní stejné.

CommDTM bylo testováno na zařízeních TF-211 a Prosonic T umístěných v řídícím systému. Veškerá komunikace probíhala v těchto případech korektně. Pro účely testování obecného grafického rozhraní byl firmou Siemens zapůjčen teploměr profilu 3.0. Po umístění do řídícího systému se však nepodařilo inicializovat mezi CommDTM a zařízením komunikační kanál, který by pak mohl DeviceDTM využít pro komunikaci. Důvod, proč není tento kanál inicializován, se před odevzdáním diplomové práce nepodařilo zjistit.

Záměrem při tvorbě obou modulů bylo vytvořit DTM, které by katedra řídící techniky mohla využít při výuce. Tomu se podřizuje i uživatelské rozhraní DeviceDTM, které je spíše zaměřeno na demonstraci struktury zařízení pracujícím na profilu 3.0, než jako DTM s profesionálními funkcemi. Přesto však toto DTM zvládá všechny potřebné funkce, které jsou od DTM vyžadovány.

# 12. Použitá literatura


[1] FDT-JOINT INTEREST GROUP. *FDT Interface Specification (Version 1.2.1)*. 2005.

[2] PROFIBUS NUTZERORGANISATION. *Profile for Process Control Devices (Version V 3.01)*. 2004.

[3] PROFIBUS NUTZERORGANISATION. *Profile for Process Control Devices (Version 3.0)*. 1999.

[4] FDT-JOINT INTEREST GROUP. *Field Device Tool for PROFIBUS (Version 1.0) : annex to FDT Specification*. 2005. Based on FDT Specification Version 1.2.1.

[5] PROFIBUS NUTZERORGANISATION. *Device Integration : GSD, EDD, FDT*. Device Integration in Field Bus Technology.

[6] PROFIBUS NUTZERORGANISATION. *DTM Styleguide (Version 1.1)*. 2000. Guideline for the implementation of Device Type Managers (DTMs) for field devices.

[7] M&M SOFTWARE; ZIMMERMAN, P. *dtmManager (Version: 1.7) : dtmMANAGER Tutorial and Programmer's Guide*. 2006.

[8] M&M SOFTWARE. *dtmManager (Version: 1.8) : dtmMANAGER Tutorial and Programmer's Guide*. 2006.

[9] TREFZER, A. *dtmManager : FDT 1.2.1 Upgrade Guide*. 2006. Version 0.1.

[10] SIEMENS. *SIMATIC NET : DPC2 Programming Interface.*

[11] SMOLÍK, P. *Průmyslová sběrnice Profibus DP Extended*. Diplomová práce, ČVUT FEL, 2001.

[12] HLAVÁČ, D. *Přístup ke sběrnici Profibus PA přes internet*. Diplomová práce, ČVUT FEL, 2003.

[13] KAB. Volba správného řešení – FDT/DTM versus EDD. *Automatizace* [online]. 2005, *2*, 79. [cit. 20-02-06]. Available from www: <http://www.automatizace.cz/article.php?a=526>.

[14] KŘENA, M. FDT/DTM – standard budoucnosti. *Automatizace* [online]. 2004, *12*, 728. [cit. 20-02-06]. Available from www: <http://www.automatizace.cz/article.php?a=431>.


## Internetové adresy


| | | |
|---|---|---|
| [15] | www.profibus.cz | PROFIBUS v České republice |
| [16] | www.profibus.com | PROFIBUS mezinárodní |
| [17] | www.siemens.com | Stránky Siemens |
| [18] | www.abb.com | Stránky ABB |
| [19] | www.mm-software.com | Stránky M&M Software |
| [20] | www.endress.com | Stránky Endress+Hauser |

# 13. Seznam příloh

**Příloha A** **Parametry fyzického bloku (profil 3.0)**

**Příloha B** **Parametry převodního bloku (profil 3.0)**

**Příloha C** **Parametry funkčního bloku (profil 3.0)**

# Příloha A – Parametry fyzického bloku (profil 3.0)

| Parameter | Description |
|---|---|
| DEVICE_CERTIFICATION | Certifications of the field device, e.g. EX certification. |
| DESCRIPTOR | User-definable text (a string) to describe the device within the application. |
| DEVICE_INSTAL_DATE | Date of installation of the device. |
| DEVICE_MESSAGE | User-definable MESSAGE (a string) to describe the device within the application or in the plant. |
| DEVICE_ID | Manufacturer specific identification of the device. |
| DEVICE_MAN_ID | Identification code of the manufacturer of the field device. |
| DEVICE_SER_NUM | Serial number of the field device. |
| DIAGNOSIS | Detailed information of the device, bitwize coded. More than one message possible at once. If MSB of byte 4 is set to 1 than more diagnose information is available in the DIAGNOSIS_EXTENSION parameter. |
| DIAGNOSIS_MASK | Definition of supported DIAGNOSIS information-bits.<br>0: not supported<br>1: supported |
| DIAGNOSIS_MASK_ EXTENSION | Definition of supported DIAGNOSIS_EXTENSION information-bits.<br>0: not supported<br>1: supported |
| FACTORY_RESET | Coding:<br>1: (mandatory) is the command for resetting device for default values. The setting of the bus address is not affected.<br>2: (optional) is the command for resetting informational device parameters to default values. Parameters with Reset Class characteristic “informational” are defined within the parameter attribute table of each block. The setting of the bus address is not affected.<br>3: (optional) is the command for resetting device parameters with Reset Class characteristic “functional” to default values. The setting of the bus address is not affected.<br>3 – 2505: reserved<br>2506: (optional) is the command for warmstart of the device. All parametrisation remains unchanged.<br>2507 – 2711: reserved<br>2712: (optional) The bus address to its default address; other parametrisation remains unchanged. The bus address is changed immediately regardless if the device is in cyclic data transfer state. The reset is not suspended up to a subsequent power cycle/warmstart.<br>2713 – 32767: reserved<br>32768 – 65535: manufacturer specific<br>Manufacturing specific commands for other reset results are possible.<br>The Ident_Number_Selector parameter isn’t effected by the Factory_Reset.<br>Note: Address manipulation by local display is not in the scope of this profile. |
| HARDWARE_REVISION | Revision-number of the hardware of the field device. |
| SOFTWARE_REVISION | Revision-number of the software of the field device. |
| LOCAL_OP_ENA | Local operation enable.<br>0: disabled (Local operation not allowed, i.e. change of FB MODE from host device only)<br>1: enabled (Local operation is allowed)<br>The operation of the host has higher priority then the local terminal one.<br>If communication fails for a time greater 30 sec, local operation will be enabled automatically. Communication failure is defined here as absence of cyclic and acyclic communication for the specified time period. If LOCAL_OP_ENA parameter is equal 0 (disabled) and the communication is working again, then the device switch back to remote operation. |
| WRITE_LOCKING | Software write protection.<br>0: acyclic write service of all parameter are refused, except WRITE_LOCKING and the TAB_ENTRY parameter of the Linerarisation table, i.e. access is denied.<br>1 – 2456: reserved<br>2457: is the default value and means all writeable parameters of a device are writeable.<br>2458 – 32767: reserved |

| | |
|---|---|
| | 32768 – 65535: manufacturer specific |
| HW_WRITE_ PROTECTION | Indicates the position of a write blocking mechanism (e.g. hardware jumper) which protects acyclic write access to writeable parameters of a device.<br>0: Unprotected (mandatory)<br>1: Protected, manual operation permitted (optional)<br>Acyclic write access to all parameter is refused (write access is denied) except the TAB_ENTRY parameter of the Linearisation table and parameters TARGET_MODE and OUT/OUT_D (only valid for AO and DO).<br>2: Protected, no manual operation (optional)<br>Acyclic write access to all parameter is refused (write access is denied) except the TAB_ENTRY parameter of the Linearisation table.<br>3 – 127: reserved<br>128 – 255: manufacturer specific<br>A device may support code 1 and/or 2. |

| Relative Index | Parameter Name | Object Type | Data Type | Store | Size | Access | Parameter Usage / Kind of Transport | Reset Class | Default Value | Mandatory Optional (Class A,B) |
|---|---|---|---|---|---|---|---|---|---|---|
| ... Standard Parameters see General Requirements | | | | | | | | | | |
| | | | | | | | | | | |
| Additional Physical Block Parameters | | | | | | | | | | |
| 8 | SOFTWARE_REVISION | Simple | VisibleString | Cst | 16 | r | C/a | - | - | M |
| 9 | HARDWARE_REVISION | Simple | VisibleString | Cst | 16 | r | C/a | - | - | M |
| 10 | DEVICE_MAN_ID | Simple | Unsigned16 | Cst | 2 | r | C/a | - | - | M |
| 11 | DEVICE_ID | Simple | VisibleString | Cst | 16 | r | C/a | - | - | M |
| 12 | DEVICE_SER_NUM | Simple | VisibleString | Cst | 16 | r | C/a | - | - | M |
| 13 | DIAGNOSIS | Simple | OctetString byte4,MSB=1 more diag available | D | 4 | r | C/a | - | - | M |
| 14 | DIAGNOSIS_EXTENSION | Simple | Octetstring | D | 6 | r | C/a | - | - | O |
| 15 | DIAGNOSIS_MASK | Simple | Octetstring | Cst | 4 | r | C/a | - | - | M |
| 16 | DIAGNOSIS_MASK_ EXTENSION | Simple | Octetstring | Cst | 6 | r | C/a | - | - | O |
| 17 | DEVICE_CERTIFICATION | Simple | VisibleString | Cst | 32 | r | C/a | - | - | O |
| 18 | WRITE_LOCKING | Simple | Unsigned16 | N | 2 | r,w | C/a | F | - | O |
| 19 | FACTORY_RESET | Simple | Unsigned16 | S | 2 | r,w | C/a | F | - | O |
| 20 | DESCRIPTOR | Simple | OctetString | S | 32 | r,w | C/a | I | - | O |
| 21 | DEVICE_MESSAGE | Simple | OctetString | S | 32 | r,w | C/a | I | - | O |
| 22 | DEVICE_INSTAL_DATE | Simple | OctetString | S | 16 | r,w | C/a | I | - | O |
| 23 | LOCAL_OP_ENA | Simple | Unsigned8 | N | 1 | r,w | C/a | F | 1 | O |
| 24 | IDENT_NUMBER_ SELECTOR | Simple | Unsigned8 | S | 1 | r,w | C/a | - | - | M (B) |
| 25 | HW_WRITE_ PROTECTION | Simple | Unsigned8 | D | 1 | r | C/a | - | - | O |
| 26 | FEATURE | Record | DS-68 | N | 8 | R | C/a | - | - | M (for Revision 3.01) |
| 27-32 | Reserved by PNO | | | | | | | | | |

# Příloha B – Parametry převodního bloku (profil 3.0)

| Parameter | Description |
|---|---|
| BIAS_1 | Bias that can be algebraically added to the process value of channel 1.<br>The unit of BIAS_1 is the PRIMARY_VALUE_UNIT. |
| BIAS_2 | Bias that can be algebraically added to the process value of channel 2.<br>The unit of BIAS_2 is the PRIMARY_VALUE_UNIT. |
| INPUT_FAULT_GEN | Input malfunction: Diagnosis object for errors that concerns all values<br>0: device OK<br>Bit 0 Rj error<br>Bit 1 Hardware error<br>Bit 2 - 4 reserved<br>Bit 5 - 7 manufacturer specific |
| INPUT_FAULT_1 | Input malfunction: Diagnosis object for errors that concern SV_1<br>0: Input OK<br>Bit 0 underrange<br>Bit 1 overrange<br>Bit 2 lead breakage<br>Bit 3 short circuit<br>Bit 4 - 5 reserved<br>Bit 6 - 7 manufacturer specific |
| INPUT_FAULT_2 | Input malfunction: Diagnosis object for errors that concern SV_2<br>Bit definition see INPUT_FAULT_1 |
| INPUT_RANGE | Electrical input range and mode. The ranges are manufacturer specific but range n is smaller than range n+1 if more than one range is supported for one input mode (e.g. range1 = 0...400W, range2 = 0...4kW).<br>INPUT_RANGE is equal for channel 1 and 2.<br>Coding (other codes are reserved):<br>0: mV range 1<br>1: mV range 2<br>...<br>9: mV range 10<br>128: W range 1<br>129: W range 2<br>...<br>137: W range 10<br>192: mA range 1<br>193: mA range 2<br>...<br>201: mA range 10<br>240: manufacturer specific<br>...<br>249: manufacturer specific<br>250: not used<br>251: none<br>252: unknown<br>253: special<br>Remark: When using codes 240..249 (manufacturer specific) interchangeability is not possible. |
| LIN_TYPE | Select the type of sensor (Code) for Thermocouples, Rtd, Pyrometers or linear. |
| UPPER_SENSOR_LIMIT | Physical upper limit function of the sensor (e.g. Pt 100 = 850°C) and input range.<br>In the case of multichannel measurements (e.g. differential measurement) the meaning of UPPER_SENSOR_LIMIT is the limit of one channel and not the calculated limit of both channels.<br>The unit of UPPER_SENSOR_LIMIT is the PRIMARY_VALUE_UNIT. |
| LOWER_SENSOR_LIMIT | Physical lower limit function of the sensor (e.g. Pt 100 = -200°C) and input range.<br>In the case of multichannel measurements (e.g. differential measurement) the meaning of LOWER_SENSOR_LIMIT is the limit of one channel and not the calculated limit of both channels. |

| | |
|---|---|
| | The unit of LOWER_SENSOR_LIMIT is the PRIMARY_VALUE_UNIT. |
| MAX_SENSOR_VALUE_1 | Holds the maximum SECONDARY_VALUE_1. The unit is defined in SECONDARY_VALUE_1. |
| MIN_SENSOR_VALUE_1 | Holds the minimum SECONDARY_VALUE_1. The unit is defined in SECONDARY_VALUE_1. |
| MAX_SENSOR_VALUE_2 | See. MAX_SENSOR_VALUE_1 |
| MIN_SENSOR_VALUE_2 | See. MIN_SENSOR_VALUE_1 |
| PRIMARY_VALUE | Selects the unit code of the PRIMARY_VALUE and other values.<br>Minimum set of unit codes:<br>1000: K (Kelvin)<br>1001: °C (degree Celsius)<br>1002: °F (degree Fahrenheit)<br>1003: Rk (Rankine)<br>Electrical units must be supported according to the supported INPUT_RANGE codes (for LIN_TYPE=0). |
| SECONDARY_VALUE_1 (SV_1) | Process value connected to channel 1 and corrected by BIAS_1. The unit of SECONDARY_VALUE_1 is the PRIMARY_VALUE_UNIT. |
| SECONDARY_VALUE_2 (SV_2) | Process value connected to channel 2 and corrected by BIAS_2. The unit of SECONDARY_VALUE_2 is the PRIMARY_VALUE_UNIT. |

| Relative Index | Parameter Name | Object Type | Data Type | Store | Size | Access | Parameter Usage / Kind of Transport | Reset Class | Default Value | Download Order | Mandatory Optional (Class A,B) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ... Standard Parameters see General Requirements | | | | | | | | | | | |
| | | | | | | | | | | | |
| Additional Temperature Transducer Block Parameters | | | | | | | | | | | |
| 8 | PRIMARY_VALUE | Record | 101 | D | 5 | r | C/a | - | | | M |
| 9 | PRIMARY_VALUE_UNIT | Simple | Unsigned16 | S | 2 | r,w | C/a | F | | 2 | M |
| 10 | SECONDARY_VALUE_1 | Record | 101 | D | 5 | r | C/a | - | | | M |
| 11 | SECONDARY_VALUE_2 | Record | 101 | D | 5 | r | C/a | - | | | O |
| 12 | SENSOR_MEAS_TYPE | Simple | Unsigned8 | S | 1 | r,w | C/a | F | | 3 | M |
| 13 | INPUT_RANGE | Simple | Unsigned8 | S | 1 | r,w | C/a | F | | 4 | M |
| 14 | LIN_TYPE | See 3.8.2, Table 55. | | | | | | | | 1 | M |
| 19 | BIAS_1 | Simple | Float | S | 4 | r,w | C/a | F | 0.0 | 5 | M |
| 20 | BIAS_2 | Simple | Float | S | 4 | r,w | C/a | F | 0.0 | | O |
| 21 | UPPER_SENSOR_LIMIT | Simple | Float | N | 4 | r | C/a | - | | | M |
| 22 | LOWER_SENSOR_LIMIT | Simple | Float | N | 4 | r | C/a | - | | | M |
| 24 | INPUT_FAULT_GEN | Simple | Unsigned8 | D | 1 | r | C/a | - | | | M |
| 25 | INPUT_FAULT_1 | Simple | Unsigned8 | D | 1 | r | C/a | - | | | M |
| 26 | INPUT_FAULT_2 | Simple | Unsigned8 | D | 1 | r | C/a | - | | | O |
| 27 | SENSOR_WIRE_ CHECK_1 | Simple | Unsigned8 | S | 1 | r,w | C/a | F | | | O |
| 28 | SENSOR_WIRE_ CHECK_2 | Simple | Unsigned8 | S | 1 | r,w | C/a | F | | | O |
| 29 | MAX_SENSOR_VALUE_1 | Simple | Float | N | 4 | r,w | C/a | I | | | O |
| 30 | MIN_SENSOR_VALUE_1 | Simple | Float | N | 4 | r,w | C/a | I | | | O |
| 31 | MAX_SENSOR_VALUE_2 | Simple | Float | N | 4 | r,w | C/a | I | | | O |
| 32 | MIN_SENSOR_VALUE_2 | Simple | Float | N | 4 | r,w | C/a | I | | | O |

# Příloha C – Parametry funkčního bloku (profil 3.0)

| Parameter | Description |
|---|---|
| OUT | The Function Block parameter OUT contains the current measurement value in a vendor specific or configuration adjusted engineering unit and the belonging state in AUTO MODE. The Function Block parameter OUT contains the value and status set by an operator in MAN MODE. |
| PV_SCALE | Conversion of the Process Variable into percent using the high and low scale values. The engineering unit of PV_SCALE high and low scale values are directly related to the PV_UNIT of the configured Transducer Block (configured via Channel parameter). The PV_SCALE high and low scale values follow the changes of the PV_UNIT of the related Transducer Block automatically, i.e. a change of the Transducer Block PV_Unit causes no bump at OUT from AI. There are exceptions possible where the bumb is required such as cleaning of analysers. |
| PV_FTIME | Filter time of the Process Variable.<br>The Function Block parameter PV_FTIME contains the time constant for the rise time of the FB output up to a value of 63,21 % resulted from a jump on the input (PT1 filter). The engineering unit of the parameter is second. |
| ALARM_HYS | Hysteresis<br>Within the scope of the PROFIBUS-PA specification for transmitters there are functions for the monitoring of limit violation (off-limit conditions) of adjustable limits.<br>Maybe the value of one process variable is just the same as the value of a limit and the variable fluctuates around the limit it will occur a lot of limit violations. That triggers a lot of messages; so it must be possible to trigger messages only after crossing an adjustable hysteresis. The sensitivity of triggering of the alarm messages is adjustable. The value of the hysteresis is fixed in ALARM_HYS and is the same for the parameters HI_HI_LIM, HI_LIM, LO_LIM and LO_LO_LIM. The hysteresis is expressed as value below high limit and above low limit in the engineering unit of xx_LIM. |
| HI_HI_LIM | Value for upper limit of alarms<br>Upper limit value for alarms with the same engineering unit of the OUT parameter. If the measured variable is equal or higher than the upper limit value the State Bits in the State Byte of OUT and in the FB parameter ALARM_SUM have to change to 1. The unit of this parameter is the same like the OUT one. |
| HI_LIM | Value for upper limit of warnings<br>Upper limit value for warnings with the same engineering unit of the OUT parameter. If the measured variable is equal or higher than the upper limit value, the State Bits in the State Byte of OUT and in the FB parameter ALARM_SUM have to change to 1. The unit of this parameter is the same like the OUT one . |
| LO_LIM | Value for lower limit of warnings<br>Lower limit value for warnings with the same engineering unit of the OUT parameter. If the measured variable is equal to or lower than the lower limit value, the State Bits in the State Byte of OUT and in the FB parameter ALARM_SUM have to change to 1. The unit of this parameter is the same like the OUT one. |
| LO_LO_LIM | Value for the lower limit of alarms<br>Lower limit value for alarms with the same engineering unit of the OUT parameter. If the measured variable is equal to or lower than the lower limit value, the State Bits in the State Byte of OUT and in the FB parameter ALARM_SUM have to change to 1. The unit of this parameter is the same like the OUT one. |
| HI_HI_ALM | State of the upper limit of alarms<br>This parameter contains the state of the upper limit of an alarm and the related time stamp. The time stamp expresses the time the measured variable has been equal or higher than the upper limit of the alarm. Devices without clock use the beginning of the PROFIBUS-PA time (1st January 1984) as time stamp. |
| HI_ALM | State of the upper limit of warnings<br>This parameter contains the state of the upper limit of a warning and the related time stamp. The time stamp expresses the time the measured variable has been |

| | |
|---|---|
| | equal or higher than the upper limit of the warning. Devices without clock use the beginning of the PROFIBUS-PA time (1st January 1984) as time stamp. |
| LO_ALM | State of the lower limit of warnings<br>This parameter contains the state of the lower limit of a warning and the related time stamp. The time stamp expresses the time at which the measured variable has been equal to or lower than the lower limit of the warning. Devices without clock use the beginning of the PROFIBUS-PA time (1st January 1984) as time stamp. |
| LO_LO_ALM | State of the lower limit of alarms<br>This parameter contains the state of the lower limit of an alarm and the related time stamp. The time stamp expresses the time at which the measured variable has been equal to or lower than the lower limit of the alarm. Devices without clock use the beginning of the PROFIBUS-PA time (1st January 1984) as time stamp. |
| OUT_UNIT_TEXT | If a specific unit of OUT parameter is not in the code list (see General Requirement) the user has the possibility to write the specific text in this parameter. The unit code is then equal to “textual unit definition”. |

| Relative Index | Parameter Name | Object Type | Data Type | Store | Size | Access | Parameter Usage / Kind of Transport | Reset Class | Default Value | Download Order | Mandatory Optional (Class A,B) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ... Standard Parameters see General Requirements | | | | | | | | | | | |
| | | | | | | | | | | | |
| Additional Analog Input Function Block Parameters | | | | | | | | | | | |
| 10 | OUT | Record | 101 | D | 5 | r [1] | O/cyc | - | measured of the variable, state | - | M (A,B) |
| 11 | PV_SCALE | Array [2] | Float (*) | S | 8 | r,w | C/a | F | 100,0 | 1 | M (A,B) |
| 12 | OUT_SCALE | Record | DS-36 | S | 11 | r,w | C/a | F | 100,0 ,-,- | 3 | M (B) |
| 13 | LIN_TYPE | Simple | Unsigned8 | S | 1 | r,w | C/a | F | 0 | 2 [3] | M (B) |
| 14 | CHANNEL | Simple | Unsigned16 | S | 2 | r,w | C/a | F | - | - | M (B) |
| 16 | PV_FTIME | Simple | Float | S | 4 | r,w | C/a | F | 0 | - | M (A,B) |
| 17 | FSAFE_ TYPE [4] | Simple | Unsigned8 | S | 1 | r,w | C/a | F | 1 | - | O (B) |
| 18 | FSAFE_VALUE | Simple | Float | S | 4 | r,w | C/a | F | - | - | O (B) |
| 19 | ALARM_HYS | Simple | Float | S | 4 | r,w | C/a | F | 0.5 % of range | - | M (A,B) |
| 21 | HI_HI_LIM | Simple | Float | S | 4 | r,w | C/a | F | max value | 4.1 | M (A,B) |
| 23 | HI_LIM | Simple | Float | S | 4 | r,w | C/a | F | max value | 4.2 | M (A,B) |
| 25 | LO_LIM | Simple | Float | S | 4 | r,w | C/a | F | min value | 4.3 | M (A,B) |
| 27 | LO_LO_LIM | Simple | Float | S | 4 | r,w | C/a | F | min value | 4.4 | M (A,B) |
| 30 | HI_HI_ALM | Record | DS-39 | D | 16 | r | C/a | - | 0 | - | O (A,B) |
| 31 | HI_ALM | Record | DS-39 | D | 16 | r | C/a | - | 0 | - | O (A,B) |
| 32 | LO_ALM | Record | DS-39 | D | 16 | r | C/a | - | 0 | - | O (A,B) |
| 33 | LO_LO_ALM | Record | DS-39 | D | 16 | r | C/a | - | 0 | - | O (A,B) |
| 34 | SIMULATE | Record | DS-50 | S | 6 | r,w | C/a | F | dis-able | - | M (B) |
| 35 | OUT_UNIT_TEXT | Simple | OctetString | S | 16 | r,w | C/a | - | - | - | O (A,B) |
| 36-44 | Reserved by PNO | | | | | | | | | | M (A,B) |