České vysoké učení technické v Praze,
fakulta elektrotechnická,
katedra řídicí techniky

Diplomová práce

PŘÍSTUP KE SBĚRNICI PROFIBUS PA PŘES INTERNET

Dušan Hlaváč
2003

Zadávací formulář

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

Anotace

Tato diplomová práce se zabývá popisem konfiguračního systému pro zařízení na průmyslové sběrnici Profibus, specielně pro zařízení pracující na jejím rozšíření Profibus PA.

Systém obsahuje serverovou část obsluhující sběrnici, databázi pro podporu vzdálených klientů a jednoduchého webového klienta.

Konfiguraci lze provádět z jakéhokoliv počítače připojeného na internet a vybaveného prohlížečem podporujícím JavaScript nebo přímo ze stanice připojené ke sběrnici Profibus, na které běží serverová část systému.

Annotation

This thesis describes the configuration system for devices on the industrial bus Profibus, especially for Profibus PA devices.

The system is divided into 3 parts – a server for communication with the bus, a database support for remote clients and a simple website client.

Primary is designed for configuration thin website client, but it is also possible to configure PA devices from the computer connected to the Profibus via server part of the system.

The client computer requirements are the connection to the internet and installed internet browser with JavaScript support.

Obsah

Zadávací formulář i
Prohlášení ii
Anotace iii
Annotation iii
Obsah iv
Tabulky vi
Obrázky vi
1 Úvod 1
2 Profibus 2
2.1 Rozdělení Profibus 2
2.2 Profibus FMS 3
2.3 Profibus DP 3
2.4 Profibus PA 3
2.4.1 Úvod 3
2.4.2 Fyzická vrstva 4
2.4.3 Rozšířená komunikace 4
2.4.4 Spojení DP a PA 5
2.5 Profily zařízení Profibus PA 5
2.5.1 Úvod 5
2.5.2 Model 5
2.5.3 Directory object 6
2.5.4 Typy bloků 8
2.5.4.1 Fyzický blok (PB) 8
2.5.4.2 Převodníkový blok (TB) 9
2.5.4.3 Funkční blok (FB) 9
3 Konfigurace zařízení Profibus PA 9
3.1 Možnosti konfigurace 9
3.2 Nástroje dodané výrobcem 10
3.3 Univerzální nástroje 10
4 Nástroje pro popis zařízení 11
4.1 Hierarchie nástrojů Profibus 11
4.2 GSD soubor 12
4.3 FDT 12
4.4 EDDL 13
5 Jazyk EDDL 13
5.1 Cíl 13
5.2 Z čeho vychází 14
5.3 Architektura 15
5.3.1 EDD source code 16
5.3.2 EDD Preprocesor 16
5.3.3 EDD Interpreter 16
5.3.4 EDD Compiler 16
5.3.5 EDD binary code 16
5.3.6 EDD Server 16
5.3.7 Možná řešení 16
5.4 Základní elementy jazyka 17
5.4.1 Víceznačnosti 18
5.4.2 Block 19

5.4.3 Variable 20
5.4.4 Menu 23
5.4.5 Command 24
5.5 Příklad popisu v UML 25
5.6 Zhodnocení 26
6 Popis konfiguračního systému 26
6.1 Požadavky a cíle 26
6.1.1 Konfigurace na dálku 27
6.1.2 Konfigurace nezávislá na typu a výrobci zařízení 27
6.2 Struktura systému 27
6.2.1 Nezávislé bloky 27
6.2.2 Použité technologie 29
6.3 Server 30
6.3.1 Schopnosti 30
6.3.2 Vnitřní uspořádání 31
6.3.3 CServer 32
6.3.3.1 CQueue 33
6.3.3.2 CDevice 34
6.3.3.3 CBlock 36
6.3.3.4 CEDDLReader 37
6.3.3.5 CVariable 38
6.3.4 Integrovaný klient 40
6.3.4.1 CVarListCtrl 41
6.3.5 Konfigurovatelnost 41
6.4 Databáze 42
6.4.1 Použitý databázový server 42
6.4.2 Výhody a omezení MySQL 43
6.4.3 Struktura databáze 43
6.5 Internetový klient 45
6.5.1 Alternativy řešení 45
6.5.1.1 Tlustý klient 45
6.5.1.2 Tenký klient 46
6.5.2 Zvolené řešení 46
6.5.3 Vlastnosti navrženého klienta 46
6.5.4 Umístění a přístupová práva 48
7 Závěr 49
8 Použitá literatura 50
9 Přílohy 51

Tabulky

Tabulka 2.1 – Hlavička Directory Objectu 7
Tabulka 2.2 – Struktura položky Composite List Directory Entries 7
Tabulka 2.3 – Struktura položky Composite Directory Entries 8
Tabulka 2.4 – Příklady adresářových struktur 8
Tabulka 5.1 – Elementy jazyka EDDL.....18
Tabulka 5.2 – Dostupné třídy pro element VARIABLE 20
Tabulka 5.3 – Datové typy definované v EDDL 22
Tabulka 6.1 – Příklad dat v souboru *block_ident.blo* pro určení bloku...37
Tabulka 6.2 – Rozpoznání bloku 42
Tabulka 6.3 - Základní uživatelé v klientské části 48

Obrázky

Obrázek 2.1 – Kód Manchester II..... 4
Obrázek 2.2 – Model zařízení Profibus PA dle profilu 6
Obrázek 2.3 – Directory object, položky a použití..... 6
Obrázek 4.1 – Oblasti použití GSD, EDD a FDT.....11
Obrázek 5.1 – Příklad zdrojového souboru v jazyce EDDL.....15
Obrázek 5.2 – Integrace EDD do systému.....17
Obrázek 5.3 – Příklad definice VARIABLE pomocí UML.....25
Obrázek 6.1 – Rozdělení systému do bloků.....28
Obrázek 6.2 – Modularita systému 29
Obrázek 6.3 – Vnitřní uspořádání tříd DPC2_Serveru 31
Obrázek 6.4 – Jednoduchá komunikace 32
Obrázek 6.5 – Složitější komunikace 33
Obrázek 6.6 – Integrovaný klient 40
Obrázek 6.7 – Struktura databáze.....44
Obrázek 6.8 – Struktura stránek internetového klienta 47

1 Úvod

Moderní průmyslové řídicí systémy jsou stále složitější a nároky na jejich schopnosti vyšší. Již několik let se nahrazuje analogový přenos měřených veličin digitálním a propojení jednotlivých spolupracujících komponent metodou každý s každým podstatně přehlednějšími a na kabeláž méně náročnými průmyslovými sběrnicemi.

První průmyslové sběrnice vznikaly nezávisle na sobě a speciálně na míru jednotlivým výrobců. Ne všechny protokoly byly otevřené či použitelné pro ostatní výrobce. Úspěšnými tedy byli jen dodavatelé kompletních řídicích systémů – tedy zařízení včetně sady programovacích, konfiguračních, vizualizačních a kontrolních nástrojů. Zákazník byl odkázán na jediného dodavatele celé technologie a na jeho variantu řešení.

Tento stav byl neudržitelný a tak pod tlakem zákazníků a menších firem začaly vznikat otevřené standardy průmyslových sběrnic s vlastnostmi orientovanými na cílovou oblast nasazení. Jedním z úspěšných a hojně používaných zejména v Evropě je Profibus[1]. V současné době existují tři varianty této sběrnice – Profibus FMS[2], Profibus DP[3] a Profibus PA[4].

Poslední a zároveň nejmladší varianta Profibusu, Profibus PA vznikla pro potřeby propojování a komunikace zařízení v nebezpečném a výbušném prostředí. Klade důraz na jiskrovou bezpečnost a proto je možné tuto sběrnici nasazovat do provozů v chemickém, petrochemickém, potravinářském a farmaceutickém průmyslu.

Dalším krokem vynuceným především zákazníky je standardizace nejenom komunikace a přenosových médií ale i zařízení samotných. Řídicí systém pak není navržen pro konkrétní provedení daného zařízení, ale pro celou skupinu obdobných zařízení. Při poruše některého prvku může být nahrazen obdobným zařízením třeba i jiného výrobce a celý systém bude bez nutnosti většího zásahu fungovat dále.

Cílem mé diplomové práce je vyvinutí konfiguračního nástroje nezávislého na obsluhovaných zařízeních. Právě díky standardizaci profilů zařízení PA je schopen spolupracovat s jakýmkoliv zařízením tento profil podporujícím. Další vlastností konfiguračního systému je jeho přístupnost přes internet.

Práce je rozdělena do následujících kapitol. Kapitola 2 popisuje sběrnici Profibus ve všech jejích variantách. Není příliš podrobná, vše již bylo

[1] PROFIBUS – PROcess FIeld BUS, otevřený standard průmyslové sběrnice pro výrobu a průmyslovou automatizaci

[2] FMS – Fieldbus Message Specification, objektově orientovaná sada zpráv pro komunikaci mezi řídicí částí sběrnice (PC, PLC, HMI…)

[3] DP – Decentralized Periphery, varianta Profibusu určená pro velmi rychlou výměnu velkého množství dat

[4] PA – Process Automation, varianta se speciální fyzickou vrstvou určená do výbušných a nebezpečných prostředí, napájena proudovou smyčkou

popsáno v [4], [5], [6] a [7]. Jsou vybrány pouze partie potřebné pro pochopení následujících částí. Kapitola 3 popisuje současné možnosti při konfiguraci zařízení na sběrnici Profibus PA. Aby byl vytvořený nástroj opravdu univerzální, potřeboval jsem existující profily popsané v odstavci 2.5 uložit ve formě čitelné pro počítač. K tomu jsem využil jazyk EDDL[5] popsaný v kapitolách 4 a 5. Jedná se o formu čitelnou i pro člověka, což značně usnadňuje porozumění a umožňuje provádět drobné zásahy do popisných souborů bez použití speciálních obslužných programů. Těžiště práce se nachází v kapitole 6. Je v ní popsán návrh, vývoj a struktura jednotlivých částí konfiguračního systému. Čeho se podařilo dosáhnout a jaké jsou možosti rozšíření systému je možné najít v 7. kapitole.

2 Profibus

Začátek vývoje průmyslové sběrnice Profibus sahá do poloviny 80. let 20. století. V té době se konsorcium firem Bosch, Klöckner & Möller, Siemens a další rozhodlo vytvořit otevřenou průmyslovou sběrnici. Nejdříve byl standard Profibus popsán v německé normě DIN 19245 (1989), v roce 1996 schválen jako Evropský standard EN 50170 a od roku 2000 je celosvětovým standardem s označením IEC 61 158.

Hlavními přínosy Profibusu, které z něj dělají vyhledávaný průmyslový komunikační standard jsou:

- rychlost,
- snadnost použití a univerzálnost,
- ekonomičnost a mnoha zákazníky vyzkoušená spolehlivost,
- nezávislost na výrobci a podpora zařízení Plug'n'play,
- otevřenost a standardizovanost.

Pokud je současně s Profibus sběrnicí použit i PROFInet, je možné jednou rodinou protokolů pokrýt celý firemní výrobní systém od řízení na nejnižší úrovni až po monitoring a kontrolu kvality výroby. Tím se značně sníží náklady na technickou podporu a kvalifikaci obsluhy.

2.1 Rozdělení Profibus

Jak jsem již zmínil v úvodu, Profibus existuje ve třech variantách, které postupně vznikaly podle potřeb technologií a odlišují se hierarchickou úrovní nasazení do řídicího systému.

[5] EDDL – Electronic Device Definition Language, člověku čitelný jazyk pro popis datových struktur a chování zařízení

- Profibus FMS (Fieldbus Message Specification)
- Profibus DP (Decentralized Periphery)
- Profibus PA (Process Automation)

2.2 Profibus FMS

Profibus FMS je historicky nejstarší varianta Profibusu. Nabízí objektově orientovaný systém komunikačních rámců schopný fungovat v heterogenním prostředí. Klade důraz na spolehlivost a komfort při komunikaci. To je však na úkor rychlosti a proto je tato varianta určena spíše pro komunikaci na vyšší úrovni řízení – tedy mezi PC, PLC[6], HMI[7], atd.

V dnešní době je ve většině případů nahrazován mnohem pohodlnějším a mezi programátory známějším Industrial Ethernetem.

2.3 Profibus DP

Jedná se o rychlý komunikační standard pracující na přenosovém médiu dle RS485 (kroucená dvoulinka) nebo na optickém vlákně. V závislosti na rozsáhlosti a složitosti sítě je schopen komunikovat až rychlostí 12 Mb/s.

Je určený pro přenos dat na nejnižší úrovni ve značném množství. Nejčastěji se používá u zařízení s decentralizovanými periferiemi a nahrazuje tak původní paralelní přenosy procesních hodnot. Vzhledem ke své rychlosti je také využíván pro řízení v reálném čase.

2.4 Profibus PA

2.4.1 Úvod

Jedná se o nejmladší variantu Profibusu. Vznikl jako rozšíření varianty DP pro speciální a náročná prostředí. Používá se hojně v místech, kde je potřeba zajistit jiskrovou bezpečnost (Ex) jako jsou potravinářský, chemický, farmaceutický nebo petrochemický průmysl.

Zařízení Profibus PA nahrazují původní zařízení s analogovým přenosem měřené veličiny (4-20mA). Nejen že hodnotu přenášejí jen v digitální formě, jsou schopna jí ještě před předáním řídicímu systému předzpracovat a dodávat s mnoha dalšími vlastnostmi.

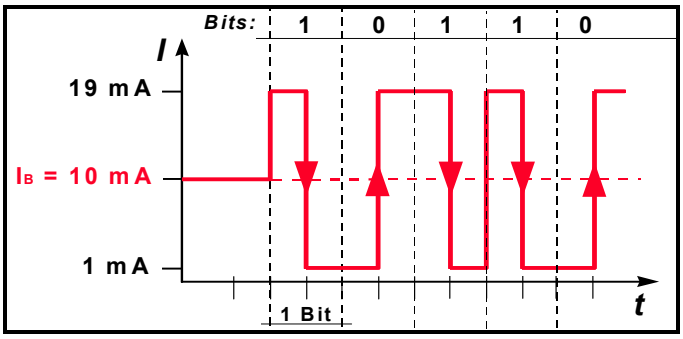
Surová nezpracovaná data jsou tak nahrazena hodnotou s jednotkami, počtem platných desetinných míst, linearizovanou či jinak numericky upravenou a s ověřenou platností.

[6] PLC – Programmable Logic Controller, logické řídicí automaty používané nejčastěji na úrovni technologických procesů

[7] HMI – Human Machine Interface, operátorská stanice pro dohled a parametrizaci řídicího systému

2.4.2 Fyzická vrstva

Výše uvedené speciální vlastnosti bylo možné dosáhnout pomocí speciální fyzické vrstvy IEC 1158-2. Jedná se o dvoudrátové vedení umožňující nejen komunikaci, ale i napájení. Je použit napěťový mód s konstantní přenosovou rychlostí 31.25 kb/s.



Obrázek 2.1 – Kód Manchester II

Ke kódování komunikace je použit Manchester II. Z Obrázek 2.1 je patrné, že ke změně úrovně dochází i když se nemění hodnota posílaného bitu. Tím je vždy střední hodnota signálu na sběrnici

$I_s = (I_{max} + I_{min}) / 2 = 10mA$

a tento proud je použit k napájení jednotlivých slave zařízení.

2.4.3 Rozšířená komunikace

Na sběrnici Profibus (ať již DP nebo PA) mohou přistupovat tyto tři typy zařízení:

- **Master třídy 1 (DPMC1)**
  Zařízení realizuje cyklickou komunikaci se zařízeními slave, je většinou reprezentováno pomocí PLC, někdy také PC, CNC… Využívá se k běžnému řízení ostatních zařízení.

- **Master třídy 2 (DPMC2)**
  Druhý typ řídícího zařízení na sběrnici. Využívá se pro speciální komunikaci při konfiguraci, dohledu, diagnostice nebo uvádění systému do provozu. Je většinou realizováno kartou v PC a doplňkovým softwarovým vybavením.

- **Slave**
  Ostatní periferní zařízení s funkcí pořizování dat nebo vykonávání akčních zásahů.

Protože zařízení Profibus PA jsou podstatně komplikovanější než původní DP a množství nastavitelných položek je značně větší, je použita rozšířená komunikace DPV1 umožňující mimo jiné acyklickou komunikaci mezi DPMC1 a slave zařízením a acyklickou komunikaci mezi DPMC2 a slave zařízením.

2.4.4 Spojení DP a PA

Spojení DP a PA částí sběrnice je možné pomocí dvou typů zařízení. První se označuje jako vazební člen. Jedná se o komplikovanější variantu připojení PA sběrnice. Vazební člen má adresu v obou typech sběrnic. PA sběrnice za ním má svůj vlastní adresní prostor. Výhodou je větší rozsah adres a nijak neomezovaná rychlost DP sběrnice (až 12 Mb/s).

Druhým typem zařízení je převodník PA/DP. Je značně jednodušší, pro ostatní zařízení zcela transparentní a nevytváří žádný nový adresovací prostor. Zajišťuje pouze převod komunikace mezi dvěma druhy fyzických vrstev. Nevýhodou je omezení rychlosti DP sběrnice na max. 93,75 kb/s.

2.5 Profily zařízení Profibus PA

2.5.1 Úvod

Důvod, proč profily zařízení (nejenom PA) vznikly a proč se je Profibus snaží prosazovat, je sjednotit druhy dodávaných zařízení, vytvořit skupiny se stejným chováním a v konečné fázi umožnit uživatelům a návrhářům řídicích systémů zaměňovat zařízení stejných typů. Pro každou skupinu je tedy vytvořen jeden profil, ten definuje např. chování zařízení, obsažená data a jejich formát nebo způsoby komunikace.

Výrobce pak v zájmu větší konkurenceschopnosti svých zařízení implementuje chování podle profilu a aby byla zaručena 100% shoda s profilem, nechá zařízení u PNO[8] pro tento profil certifikovat

Profily zařízení jsou definovány nad aplikační vrstvou ISO/OSI modelu.

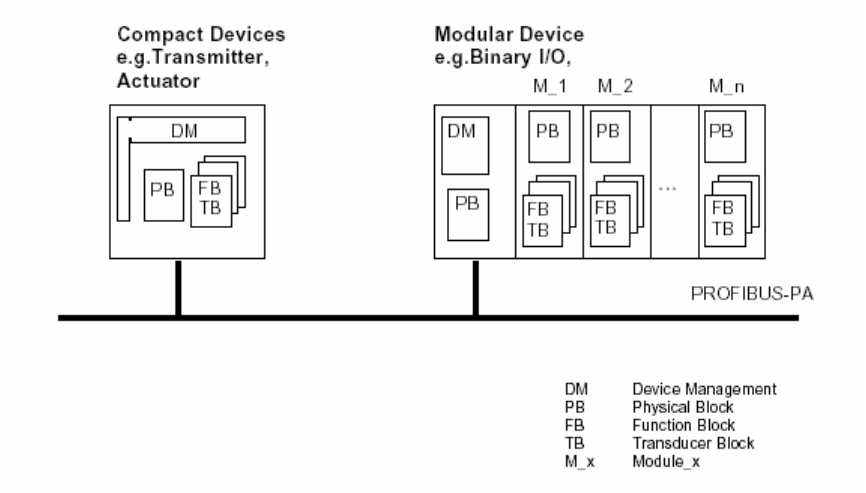
2.5.2 Model

Zařízení Profibus PA musí svou vnitřní strukturou odpovídat modelu zařízení. Ten se skládá ze správy zařízení (Device Management – DM) a bloků různých typů. Jednoduché kompaktní zařízení obsahuje vždy jeden fyzický blok (Physical Block – PB), minimálně jeden převodníkový blok (Transducer Block – TB) a min. jeden funkční blok (Function Block – FB).

Modulární zařízení se pak skládá z DM PB pro zařízení a několika samostatných modulů. Každý z modulů obsahuje svůj jeden PB a několik (opět minimálně jeden od každého) TB a FB.

Pro propojení funkčních bloků mezi sebou nebo propojení uvnitř jednoho funkčního bloku se používá spojovací objekt (Link Object). Na Obrázek 2.2 je model zařízení rozkreslen názorně v provedení jednoduchém i modulárním.

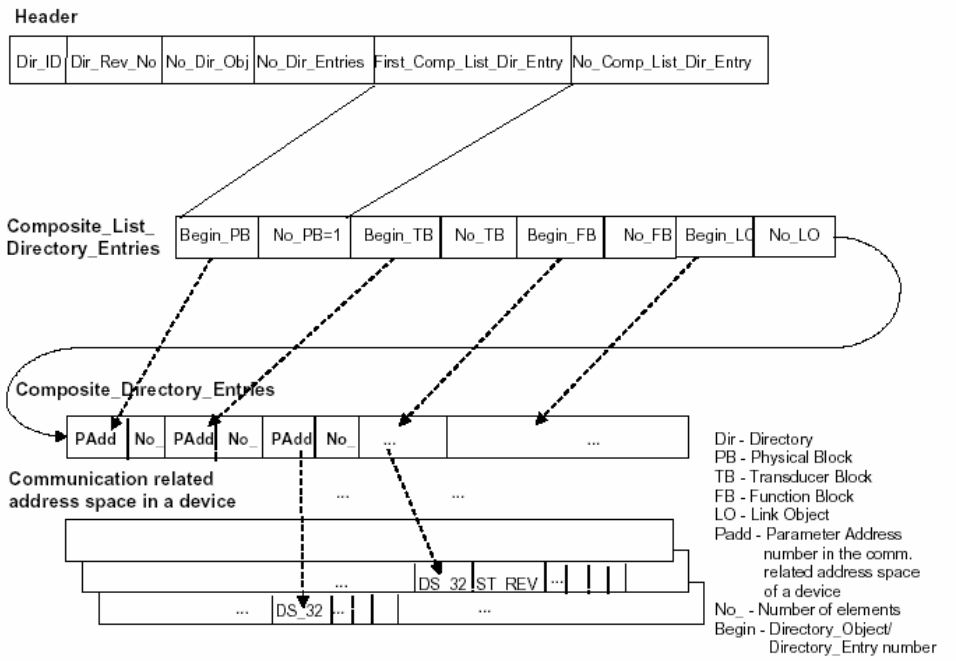
[8] PNO – Profibus NutzerOrganisation, nezisková organizace zabývající se vývojem a standardizací Profibusu, mimo jiné zajišťuje i certifikaci zařízení a produktů pro Profibus



Obrázek 2.2 – Model zařízení Profibus PA dle profilu

2.5.3 Directory object

Directory object (DO) je součástí device managementu. Používá se pro uložení pozice jednotlivých bloků zařízení. Pro každý blok je možné zjistit jeho typ, absolutní adresu první proměnné bloku a celkový počet parametrů. Na Obrázek 2.3 je vidět struktura a použití jednotlivých částí DO.



Obrázek 2.3 – Directory object, položky a použití

DO se skládá z těchto částí:

Header – hlavička, informace o velikosti adresářové struktury, obsahuje 6 dvoubytových položek, jejich názvy a významy jsou uvedeny v Tabulka 2.1.

Zkratka parametru	Velikost v bytech	Dlouhý název Význam
Dir_ID	2	Directory ID identifikace adresáře, v tomto profilu není použito
Dir_Rev_No	2	Direktory Revision Numer číslo revize
No_Dir_Obj	2	Number of Directory Objects počet položek v adresáři objektů
No_Dir_Entries	2	Total Number of Directory Entries počet položek v adresáři
First_Comp_List_Dir_Entry	2	Direktory Entry number of first Composite List Directory Entry na jakém indexu je uložen ukazatel na první blok (první je vždy PB)
No_Comp_List_Dir_Entry	2	Number of Composite List Directory Entries počet druhů bloků v zařízení (PB,TB,FB a LO)

Tabulka 2.1 – Hlavička Directory Objectu

Composite_List_Directory_Entries – obsahuje tři nebo čtyři položky (podle No_Comp_List_Dir_Entry), každá položka se vztahuje k jednomu typu bloků (PB, TB, FB, LO) a uvádí počet bloků daného typu a pozici adresářové položky prvního bloku tohoto typu. Rozepsaná struktura položky Composite_List_Directory_Entries je v Tabulka 2.2.

Název položky	Velikost v bytech	Význam
Begin_XX Index	1	Index první adresářové položky ukazující na blok typu XX
Begin_XX Offset	1	Offset první adresářové položky ukazující na blok typu XX
Num_XX	2	Počet adresářových položek typu XX

Tabulka 2.2 – Struktura položky Composite List Directory Entries

Písmena XX znamenají u první položky PB, dále pak TB, FB a poslední může nebo nemusí být uveden LO. Pořadí je pevně dané a nesmí být zaměněno.

Composite Directory Entries – položky s odkazy na začátky bloků, počet položek je dán rozdílem No_Dir_Entries a No_Comp_List_Dir_Entry. Jednotlivé části adresářové položky jsou rozepsány v Tabulka 2.3.

Název položky	Velikost v bytech	Význam
Pointer Slot	1	Slot adresy první položky v bloku
Pointer Index	1	Index adresy první položky v bloku
Num_Params	2	Počet položek uložených v bloku

Tabulka 2.3 – Struktura položky Composite Directory Entries

V Tabulka 2.4 je uvedeno několik příkladů adresářových struktur pro různé počty a typy bloků.

Typy bloků	Adresa	Uint 16	Uint 16	Uint 16	Uint 16	Uint 16	Uint 16
1x PB, 1x TB, 1x FB	1.0	00 00	00 01	00 01	00 06	00 01	00 03
	1.1	01 04	00 01	01 05	00 01	01 06	00 01
		01 141	00 25	01 51	00 90	01 06	00 35
1x PB, 3x TB, 1x FB	1.0	00 00	00 01	00 02	00 08	00 01	00 03
	1.1	02 04	00 01	02 05	00 03	02 08	00 01
	1.2	01 141	00 25	01 51	00 90	02 00	00 115
		02 115	00 139	01 16	00 35		
1x PB, 3x TB, 2x FB	1.0	00 00	00 01	00 02	00 09	00 01	00 03
	1.1	02 04	00 01	02 05	00 03	02 08	00 02
	1.2	00 16	00 25	01 51	00 90	02 76	00 115
		03 00	00 254	01 16	00 35	02 16	00 60
1x PB, 3x TB, 2x FB, 1x LO	1.0	00 00	00 01	00 02	00 11	00 01	00 04
	1.1	02 05	00 01	02 06	00 03	02 09	00 02
		02 11	00 01				
	1.2	00 16	00 25	01 51	00 90	02 76	00 115
		03 00	00 80	01 16	00 35	02 16	00 60
		03 80	00 01				

Tabulka 2.4 – Příklady adresářových struktur

2.5.4 Typy bloků

Jak již jsem zmínil v předchozích odstavcích, bloky mohou být čtyř typů: fyzický, převodníkový, funkční a linkový objekt. Každý blok má specifický význam a sobě vlastní parametry. Některé parametry mají všechny bloky společné. Jsou to základní parametry popisující blok (Block_Object), název bloku, stav bloku, seskupení bloků…

2.5.4.1 Fyzický blok (PB)

Obsahuje základní informace o zařízení, jeho výrobci, hardwaru, softwaru, verzích, sériovém čísle… Na rozdíl od ostatních bloků se fyzický

vyskytuje pouze v jediné možné variantě, všechna zařízení tedy podporují stejné proměnné v tomto bloku.

2.5.4.2 Převodníkový blok (TB)

Obsahuje informace, které se týkají připojení zařízení do procesu. Jde např. o typ senzoru, použitý materiál, fyzikální hranice atd. V dané chvíli je připojen k převodníkovému bloku jen jeden funkční blok, toto propojení se smí měnit pouze při konfiguraci zařízení. Převodníkový blok existuje v mnoha variantách dle funkce zařízení nebo měřené veličiny. Jsou definovány např. Flow (průtok), Temperature (teplota), Level (hladina), Pressure (tlak), Analyser…

2.5.4.3 Funkční blok (FB)

Provádí konečnou úpravu naměřených hodnot (např. změnu měřítka, linearizaci, offset…), které získává od převodníkového bloku. Konečné hodnoty předává po sběrnici řídicímu systému. Zařízení může obsahovat více funkčních bloků. Druhy funkčních bloků jsou např. Analog Input (AI), Analog Output (AO), Digital Input (DI) a Digital Output (DO).

Podrobnější popis typů bloků, obsažených proměnných a používaných datových typů je možné nalézt v kapitole 5 v [4]. Přesné a kompletní definice všech proměnných jsou pak popsány v [2].

3 Konfigurace zařízení Profibus PA

3.1 Možnosti konfigurace

Každé Profibus PA zařízení je třeba před uvedením do provozu nakonfigurovat. Potřebujeme tedy nástroj, který to dokáže. Nejlépe svá zařízení znají výrobci a většinou potřebný program dodávají společně se zařízením. Chceme-li konfigurovat jediné zařízení, nainstalujeme nástroj, přečteme si návod a zařízení nakonfigurujeme. Podstatně komplikovanější je nakonfigurovat více zařízení. Velkou výhodou je, pokud máme několik zařízení od jediného výrobce a společně s nimi jsme dostali nástroj schopný nakonfigurovat celou sadu zařízení. V opačném případě budeme instalovat několik programů, studovat manuály a doufat, že produkty od různých výrobců nebudou navzájem kolidovat.

Další možností je využít profilů zařízení Profibus PA a konfigurovat univerzálním nástrojem. Stačí nám k tomu ovládat vnitřní uspořádání modelu zařízení, znalost struktury bloků a pomocí asynchronní komunikace nastavíme vše potřebné. Program nepotřebuje vědět, o jaké zařízení se jedná, inteligenci mu dodává uživatel.

Takové nástroje již existují a podrobně jejich výhody a omezení proberu na příkladech v následujících odstavcích. V kapitole 6 je popsáno

další možné řešení. Jedná se o univerzální konfigurační nástroj, který je navíc informovaný. Díky znalosti profilů PA zařízení je schopen přečíst obsah zařízení a umožnit jeho konfiguraci, i když uživatel nemá o modelu zařízení a vnitřní struktuře bloků jakékoliv znalosti.

3.2 Nástroje dodané výrobcem

Jako příklad nástroje dodaného výrobcem uvádím program Commuwin II (Endress+Hauser). Jedná se o velmi zdařilý program schopný ovládat celou sadu zařízení firmy Endress+Hauser. Ukažme si na něm výhody této skupiny nástrojů.

Program ovládaná zařízení přesně zná, byl pro ně navržen. Je tedy schopen nastavit nejen parametry základní, které jsou specifikovány profilem zařízení, ale i všechny další, přidané výrobcem navíc.

Commuwin II není jen čistě konfigurační nástroj, umožňuje navíc monitoring zařízení, diagnostiku a pořizování historie dat. Získané hodnoty je schopen ukládat na disk a opětovně načítat, tisknout či vykreslovat v grafech.

Mezi nevýhody patří omezenost jen na firemní zařízení. U ostatních výrobců již nemusíme mít takové štěstí na kvalitní konfigurační nástroj schopný nastavit více než jedno zařízení.

Podrobnosti o programu Commuwin II je možné nalézt v [4] nebo přímo na stánkách výrobce [13].

3.3 Univerzální nástroje

Jako reprezentanta skupiny univerzálních nástrojů bych rád zmínil program PROtest vyvinutý institutem ifak v Magdeburgu [14]. PROtest je program pracující pod MS Windows a realizující funkce Profibus DP/DPV1 master stanice. Jeho univerzálnost spočívá v čisté implementaci synchronní a asynchronní komunikace s Profibus DP/DPV1/PA slave zařízeními. Veškerou inteligenci mu dodává jeho obsluha. Pro fyzický přístup ke sběrnici využívá kartu v počítači s rozhraním RS485.

Komunikace probíhá prostřednictvím připravených dialogových oken s částečně předvyplněnými parametry. Získaná data je možné kromě prostého zobrazení i ukládat do logovacích souborů. Navíc má program v sobě integrován skriptovací jazyk podobný jazyku C, s jehož pomocí lze spoustu operací automatizovat a značně zjednodušit. Stále je však k ovládnutí zařízení na sběrnici nutná znalost jeho chování a vnitřní struktury, popsané v kapitole 2.5.

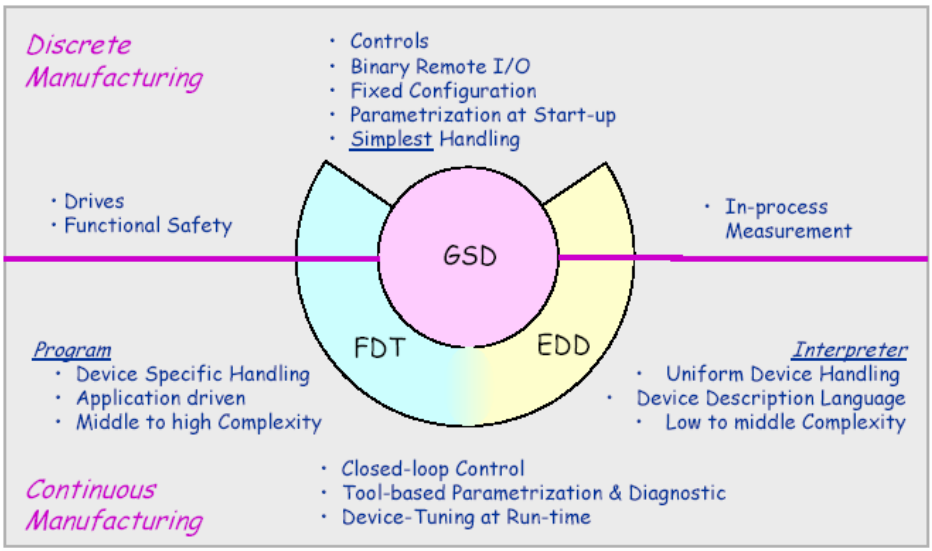
Podobným nástrojem, i když značně jednodušším, je DPC2_Analyser, který jsem vyvinul v rámci diplomového semináře. Během jeho vývoje jsem se učil používat kartu CP-5611 a k ní dodávanou podpůrnou knihovnu DPC2Lib. Program je schopen asynchronní komunikace se zařízeními na sběrnici Profibus. Dokáže obsluhovat v jeden okamžik pouze jeden komunikační kanál se zařízením. Následně umožňuje číst a zapisovat hodnoty dle zadané adresy.

Nevýhodou obou zmíněných nástrojů je jejich neinformovanost. Všechna data, se kterými se pracuje, jsou pouze surovými sadami čísel bez jakéhokoliv vysvětlení či udání datového typu. Pro uživatele důkladně neseznámeného s problematikou Profibus komunikace a zařízení je značně obtížné pochopit jejich smysl.

4 Nástroje pro popis zařízení

4.1 Hierarchie nástrojů Profibus

Již od počátku vývoje sběrnice Profibus je patrná snaha PNO[9] nabídnout uživatelům nástroje pro popis parametrů zařízení Profibus a umožnit jim tak snáze informace o zařízeních číst a předávat do SW nástrojů pro konfiguraci a správu. V poslední době integrovat jednodušší softwarové komponenty do složitějších systémů. Postupně byly vytvořeny tři navzájem na sebe navazující způsoby popisu zařízení. Oblasti nasazení jsou patrné z Obrázek 4.1.



Obrázek 4.1 – Oblasti použití GSD, EDD a FDT

[9] viz strana 5

4.2 GSD soubor

První z realizovaných standardů je tzv. GSD[10] soubor. Vznikl jako elektronická podoba technické dokumentace k zařízením Profibus. Obsahuje nejnutnější údaje o zařízení potřebné pro jeho připojení na sběrnici a následné oživení, tedy parametry jako přenosová rychlost, časování, počet I/O signálů, diagnostické zprávy…
Jedná se o textový pro člověka čitelný soubor s jasně definovanou strukturou povinných a nepovinných parametrů, jejich datových typů a limitních hodnot. Výsledkem je univerzální popisný nástroj pro možnou konfiguraci zařízení bez ohledu na jeho typ a výrobce. S nově vznikajícími zařízeními se rozšiřují i parametry GSD souborů, základní sada hodnot však zůstává zachována.

Za existenci, funkčnost a správnost GSD souboru je zodpovědný výrobce zařízení. Pokud se vyskytnou nějaké problémy, existují kontrolní nástroje na GSD soubory a nebo je možné nouzově použít tzv. minimální GSD soubor, popsaný např. v [7].

4.3 FDT

Rozhraní FDT[11] představuje podstatně složitější popis zařízení než je GSD soubor. Není již textovým člověku běžně čitelným souborem ale spíše ovladačem specifickým pro to které zařízení. Cílem je poskytnout tvůrcům diagnostických, vývojových a konfiguračních programů a systémů jednotné univerzální rozhraní bez ohledu na typ zařízení a jeho výrobce. Rozhraní je navrženo tak, že poskytuje obrovský prostor pro další rozšíření, jak budou růst nároky výrobců hardwarových zařízení i softwarových nástrojů.

Přímo každému zařízení by měl jeho výrobce vytvořit na míru softwarový nástroj zvaný DTM[12]. Může využít všechny výhody a specifika svého zařízení, pro nějž program píše. Na druhé straně musí být schopen s okolním světem (a tedy i s ostatními programy či např. internetovými prohlížeči se schopností používat FDT) komunikovat pomocí standardní sady funkcí FDT rozhraní. Rozhraní je natolik univerzální, že nemusí nutně spojovat zařízení Profibus, ale je na sběrnici nezávislé. Je možné jej zakomponovat i do heterogenních řídicích systémů s několika typy sběrnic.

Univerzálnost, škálovatelnost a otevřenost dalším rozšířením je zaručena pomocí technologií, které toto umožňují. Nejsou omezeny pevně nadefinovanou sadou parametrů a funkcí, nýbrž struktura dat a nabízené operace jsou přenášeny společně s daty vlastními. Implementace FDT

[10] GSD – z německého Gerätestammdaten, textový soubor obsahující základní informace o slave zařízení potřebné pro konfiguraci zařízení master (přenosová rychlost, počet I/O vstupů…)

[11] FDT – Field Device Tool, rozhraní pro komunikaci a přenos dat mezi DTM a aplikací s uživatelským rozhraním

[12] DTM – Device Type Manager, ovladač zařízení poskytující rozhraní FDT

systému má proto strukturu klient-server, využívá první vrstvu Microsoft COM[13] rozhraní a výměna dat probíhá výhradně pomocí XML[14].

4.4 EDDL

EDDL svojí složitostí stojí někde mezi primitivním GSD souborem a zcela komplexní a pouze pro počítač srozumitelnou kombinací FDT rozhraní a DTM programu. Nesnaží se zařízení zcela schovat za sadu univerzálních funkcí a data s popsaným formátem.

Jeho cílem je umožnit jedinému komunikačnímu nástroji se základní znalostí prostředí např. Profibus získat podrobné informace o struktuře a chování zařízení a tak nahradit celou sadu programů, z nichž každý je schopen spolupracovat jen se zařízením, pro které byl vyvinut.

Pomocí jazyka EDDL se vytvoří tzv. EDD[15] soubor (stále ještě pro člověka čitelný a snadno upravitelný), popisující jak zařízení konfigurovat, obsluhovat a co zařízení dokáže. EDDL je schopen popsat syntaktiku (formát) a sémantiku (význam) datových struktur zařízení, jeho chování a s tím související i potřebné uživatelské rozhraní. Stačí aby cílový program obsahoval EDDI – tedy interpreter jazyka EDDL a je schopen ovládat jakékoliv zařízení tímto jazykem popsané bez ohledu na jeho typ a výrobce. Opět se jedná o nástroj univerzální a použitelný i pro jiné průmyslové sběrnice, nejen Profibus.

5 Jazyk EDDL

5.1 Cíl

Před vznikem jazyka EDDL bylo pro konfiguraci systému nutně zapotřebí stejného počtu konfiguračních nástrojů jako bylo typů zařízení v systému integrovaných. Při použití konfiguračního souboru EDD v jazyce EDDL se celá konfigurace výrazně zjednoduší.

Výrobce zařízení nemusí vyvíjet konfigurační program svému zařízení na míru, nýbrž dodá jen popis zařízení prostřednictvím EDD souboru. Dodavatel konfiguračního nástroje pouze implementuje schopnost

[13] COM – Component Object Model, systém vyvinutý firmou Microsoft, umožňující programátorům vytvářet objekty, které mohou využívat libovolné aplikace, založené na tomto modelu.

[14] XML – eXtensible Markup Language, jednoduchý textový formát pro přenos dat s univerzální a snadno rozšířitelnou strukturou, vychází z SGML (ISO 8879), původně navržen pro masové publikování elektronických dokumentů, v současnosti používán pro výměnu dat různého druhu na webu i jinde.

[15] EDD – Electronic Device Description, elektronický popis zařízení běžně čitelný i bez speciálního programu

čtení EDDL jazyka a již nemusí pracně testovat svůj nástroj pro všechna zařízení. Uživateli se na konfiguračním počítači zredukuje počet programů na jeden a odstraní tak vzájemné kolize programů od různých výrobců. Výhody lze tedy shrnout do následujících bodů:

- jediný konfigurační nástroj pro všechna zařízení pro celý systém místo balíku neuniverzálních nástrojů
- chování zařízení je uloženo v EDD souboru místo v kódu softwarového nástroje
- výrobce zařízení pomocí jazyka EDDL popíše své zařízení a tvůrce konfiguračního programu musí jen přesně implementovat čtení EDD souboru
- při konfiguračně závislé změně v zařízení stačí jen opravit a dodat nový EDD soubor
- pro každý operační systém stačí jediný konfigurační nástroj
- místo neustálého dodávání novějších verzí konfiguračních programů stačí jen nahrát nové EDD soubory
- vhledem k ASCII formátu EDD souboru je vhodný k dlouhodobé archivaci – je totiž nezávislý na operačním systému, verzi programu atd.

5.2 Z čeho vychází

Jazyk EDDL je možné použít pro systém postavený na libovolné sběrnici. Při jeho návrhu se však vycházelo z následujících specifikací.

- PROFIBUS Specification (FMS, DP, PA) All normative Parts of the PROFIBUS Specification according to European Standard EN 50 170 Vol. 2. (version 1.0)
- GSD Specification for PROFIBUS-FMS Definition of the GSD-File formats for FMS (version 1.0)
- GSD Specification for PROFIBUS-DP Definition of the GSD-File formats for DP (version 3.0)
- Profile for Communication between Controllers FMS-Communication profile, specification of required services (version 1.4)
- Profile for Process Control Devices PA-Branch profile for Process Control devices (version 3.0)
- Profile for NC/RC Controllers DP profile for NC/RC Controllers (version 1.0)
- Profile for Encoders DP profile for rotary, angle and linear encoders (version 1.1)
- Profile for Variable Speed Drives FMS-/DP-Profile for electric drive technique (version 2.0)
- Profile for HMI Devices (Draft) DP-Profile for Human Machine Interface devices (version 1.0)
- Profile for Failsafe with PROFIBUS (Draft) DP-Profile for Safety Applications (version 1.0)

Tím je vlastně zaručeno, že pomocí EDD souboru bude možné popsat libovolné zařízení Profibus. EDDL je vytvořen hlavně pro ně a proto se snaží výše uvedeným profilům co nejvíce přizpůsobit.

Syntaxe jazyka vychází ze specifikace ANSI C (KERNIGHAN, RITCHIE a RITCHIE) díky jeho jednoduchosti, srozumitelnosti a značné rozšířenost v řídicí technice. Rozlišují se tedy velká a malá písmena, celky patřící dohromady jsou uvozeny složenými závorkami, jednotlivé parametry se ukončují středníkem…

Pro názornost je u nejdůležitějších konstrukcí rozkreslena struktura a závislosti i pomocí jazyka UML[16]. Příklad popisu v UML je vidět na Obrázek 5.3.

5.3 Architektura

Při použití EDDL je využíváno několik nástrojů a existují také různé formy EDD popisu. Vysvětlíme si tedy nejprve pojmy.

```
/* Example file using Electronic Device Description (EDD)              */
/* Important: This file serves as an example only, it is not normative */
/* File name: example.edd                                               */
/* 3.1. The Identification                                              */
MANUFACTURER 42,
DEVICE_TYPE 42,
DEVICE_REVISION 1,
DD_REVISION 1

VARIABLE local_variable
{
    LABEL "Local Variable";
    HELP  "Help";
    CLASS  LOCAL;
    TYPE   FLOAT
    {
        DEFAULT_VALUE 30;
        MIN_VALUE 10;
        MAX_VALUE 200;
        SCALING_FACTOR 200;
        EDIT_FORMAT "5d";
        DISPLAY_FORMAT "5d";
    }
    HANDLING READ & WRITE;
    VALIDITY TRUE;
}

BLOCK BlockIdentifier1
{
    TYPE   PHYSICAL;
    NUMBER 1;
}
```

Obrázek 5.1 – Příklad zdrojového souboru v jazyce EDDL

[16] UML – Unified Modeling Language, jazyk pro názorné modelování počítačových systémů, hierarchických závislostí, vzájemné komunikace atd.

5.3.1 EDD source code

EDD source code je zdrojový soubor jazyka EDDL velmi podobný zdrojovému kódu jazyka C. Nejedná se však o programovací jazyk ale spíše o deklaraci struktur, vlastností a vzájemných interakcí. Struktura je nejlépe patrná z příkladu na Obrázek 5.1.

Z obrázku je patrné, že přestože se jedná o velmi mocný nástroj, jeho forma je stále ještě čitelná i pro člověka. Umožňuje tak celkem snadné úpravy a vylepšení bez potřeby speciálního nástroje.

5.3.2 EDD Preprocesor

Program, který pomocí direktiv `#include, #if, #ifdef, #endif` nebo `#define` je schopen selektivně zdrojový kód upravovat, slučovat více souborů dohromady a tím připravit data pro překlad či interpretaci.

Dále je schopen ze zdrojového kódu vyjmout komentáře, které se opět zapisují stejně jako v C – tedy pomocí dvou lomítek `//` je uvozen komentář do konce řádku a pomocí kombinace lomítka a hvězdičky `/* */` je zahájen resp. ukončen komentář přes více řádek.

5.3.3 EDD Interpreter

Program, který rozumí pomocí preprocesoru předzpracovanému zdrojovému souboru a je schopen jej používat. Není to nejefektivnější varianta využití elektronického popisu EDD, ale pro většinu jednodušších programů zcela postačuje.

5.3.4 EDD Compiler

Program schopný převádět předzpracovaný EDD zdrojový kód do binární podoby, kterou následně využívá EDD Server. Binární kód není ve specifikaci jazyka EDDL nijak normován, výrobce EDD Serveru si jej tedy může navrhnout podle svých představ.

5.3.5 EDD binary code

Výsledný soubor vzniklý přeložením předzpracovaného zdrojového kódu pomocí EDD Compileru. Jedná se o formát přímo použitelný jako vstup pro EDD server, norma na něj neklade žádné nároky, forma závisí čistě na tvůrci Compileru a Serveru.

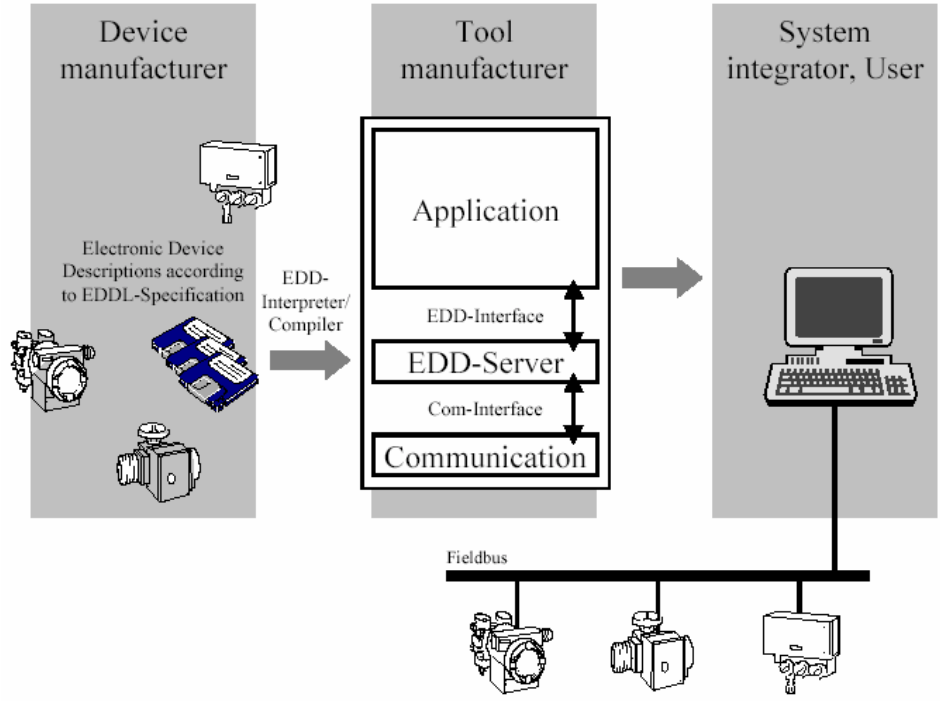
5.3.6 EDD Server

Součást výsledné komplexní aplikace pro práci s ovládaným systémem. Po načtení EDD binárního kódu je prostředníkem mezi subsystémem ovládajícím sběrnici či sběrnice a uživatelským rozhraním programu.

5.3.7 Možná řešení

Z vyjmenovaných nástrojů je možné podle složitosti navrhovaného systému vytvořit dvě různá řešení. Složitější a komplexnější přesně podle Obrázek 5.2, které se skládá z preprocesoru, kompilátoru a EDD serveru

jako výsledného vykonavatele operací specifikovaných v EDD. Protože specifikace EDDL neklade na EDD binární kód žádné nároky, je kombinace kompilátor -> binární kód -> server nerozdělitelná a musí být vyvinuta jako jeden celek. Přenositelnost binárních kódů mezi jednotlivými EDD servery je téměř nemožná, pokud nebyly navrženy pro stejný binární kód.



Obrázek 5.2 – Integrace EDD do systému

Pro jednodušší systémy není třeba implementovat složitý mechanismus překladů. Stačí použít pouze preprocesor a pomalejší ale výrazně jednodušší interpretr. Výsledná pozice v systému je stejná jakou u EDD serveru.

5.4 Základní elementy jazyka

Jazyk EDDL zná šestnáct základních elementů, ze kterých lze postavit celý popis zařízení. Jejich seznam a významy jsou uvedeny v Tabulka 5.1.

Základní element	**Význam**
Block	Popisuje relativně adresovanou skupinu proměnných
Connection	Definuje více aplikací v jednom zařízení
Variable	Popisuje datové položky obsažené v zařízení
Record	
Array	

Menu	Popisuje, jak budou data nabízena uživateli prostřednictvím hostitelského zařízení
Methode	Popisuje spuštění komplexní posloupnosti událostí, které se musí uskutečnit mezi hostitelskými a fyzickými zařízeními
Relation	Popisuje vztahy mezi proměnnými (`variable`), záznamy (`record`) a poli (`array`)
Item Array	Popisuje logické sdružování dat
Collection	
Variable List	Popisuje logické sdružování dat obsažených v zařízení, která musí být čtena a zapisována dohromady
Command	Popisuje strukturu a adresování proměnných v zařízení
Program	Upřesňuje, jak z hostitelského zařízení spustit kód běžící uvnitř fyzického zařízení
Domain	Může být použit k downloadu nebo uploadu přiměřeně velkého množství dat z nebo do zařízení
Response code	Upřesňuje kódy odpovědí závislé na aplikaci pro výše uvedené elementy jazyka EDDL

Tabulka 5.1 – Elementy jazyka EDDL

Každý element jazyka EDDL kromě `Response code` a `Relation` je možné specifikovat pro něj povolenými atributy, které se dělí na povinné a nepovinné. Některé z atributů lze ještě dále popsat pomocí speciálních sub-atributů a tak ještě zjemnit definici základního elementu. V následujících odstavcích jsou podrobně rozebrány vybrané nejdůležitější elementy jazyka EDDL. Ostatní je možno nalézt v [3].

5.4.1 Víceznačnosti

Aby se zabránilo dvoj- a víceznačnostem, nejsou povoleny opakované definice uvedené v následujících třech bodech (ke každému bodu je vždy uveden příklad):

- Základní elementy se stejným typem a identifikátorem

```
VARIABLE x
{ ... }

MENU x // NEPOVOLENO!
{ ... }
```

- Základní elementy různých typů se stejnými identifikátory

```
VARIABLE x
{
  LABEL "x";
```

```
  TYPE INTEGER;
  CLASS CONTAINED;
  CLASS CONTAINED & DYNAMIC; \\ NEPOVOLENO!
}
```

- Základní element obsahující několikrát stejný atribut

```
VARIABLE y
{
  LABEL "y";
  TYPE INTEGER
  {
        MIN_VALUE 1;
        MAX_VALUE 2;
        MIN_VALUE 3; \\ NEPOVOLENO!
  }
  CLASS CONTAINED;
  CLASS CONTAINED & DYNAMIC; \\ NEPOVOLENO!
}
```

Někdy však výrobce dodává skupinu zařízení s podobnou sadou vlastností a proměnných. Hodilo by se tak definovat základní vlastnosti a ty vždy spojit se sadou vlastností specifických jen jedinému zařízení. Jak pak řešit opakované definice proměnných se stejným názvem ale různými vlastnostmi? K tomu slouží klíčová slova `DEFINE` a `DELETE`. Standardní proměnné lze takto nahradit novými nebo jen zaměnit některé jejich vlastnosti vlastnostmi jinými. Všechny operace tohoto typu však lze provádět jen v importovaných souborech. Záměna proměnné v primárním souboru pomocí redefinice v tomtéž souboru, přestože by byla uvozena jedním z klíčových slov `REDEFINE` nebo `DELETE`, spadá do zakázaných operací uvedených na začátku tohoto odstavce.

5.4.2 Block

Klíčové slovo `BLOCK` definuje základní adresovací schéma u zařízení děleného na bloky. Každý blok musí být pojmenován a toto jméno je dále používáno, pokud se na něj jiné definice odkazují.

Dále je nutné u každého bloku uvést dva povinné parametry oddělené středníkem. Jsou to typ a číslo. Typ musí být jedním ze tří v Profibus zařízeních používaných – tedy fyzický (`PHYSICAL`), funkční (`FUNCTION`) nebo převodníkový (`TRANSDUCER`). Číslo pak odpovídá pořadovému číslu daného bloku mezi bloky stejného typu. U zařízení s jediným fyzickým, funkčním a převodníkovým blokem je každý označen jako číslo jedna. Zařízení s jedním fyzickým, dvěmi funkčními, dvěmi převodníkovými bloky by mělo očíslování bloků viz. následující příklad.

```
BLOCK PhBl1
{
   TYPE PHYSICAL;
   NUMBER 1;
}
BLOCK FuBl1
{
   TYPE FUNCTION;
   NUMBER 1;
}
BLOCK FuBl2
{
   TYPE FUNCTION;
   NUMBER 2;
}
BLOCK TrBl1
{
   TYPE TRANSDUCER;
   NUMBER 1;
}
BLOCK TrBl2
{
   TYPE TRANSDUCER;
   NUMBER 2;
}
```

5.4.3 Variable

Pomocí klíčového slova VARIABLE se definují jednotlivé datové položky uložené v zařízeních. Stejně jako u bloku musí každá proměnná mít své jedinečné jméno, které je následně používáno pro odkazování v dalších definicích. Povinné parametry jsou třída (CLASS), typ (TYPE) a popisek (LABEL).

Třída označuje, jak se daná položka bude uchovávat v hostitelském zařízení a může nabývat jedné nebo více z následujících hodnot spojených znakem &. Dostupné jsou tyto třídy:

Class type	Význam
INPUT	proměnná používaná jako výstupní z jiného bloku
OUTPUT	proměnná používaná jako vstupní pro jiný blok
CONTAINED	proměnná obsažená uvnitř bloku, na níž se nelze odkazovat z jiného bloku
DYNAMIC	proměnná pozměňovaná zařízením bez vnějšího podnětu přijatého po sběrnici
DIAGNOSTIC	proměnná obsahující diagnostickou informaci
SERVICE	proměnná, která je součástí servisního algoritmu
OPERATE	proměnná pozměňující chování bloku
ALARM	proměnná obsahující limitní hodnotu spouštějící alarm
TUNE	proměnná určená k ladění algoritmu uvnitř bloku
LOCAL	proměnná používaná pouze v hostitelském zařízení a nemá reprezentaci v žádném fyzickém zařízení

Tabulka 5.2 – Dostupné třídy pro element VARIABLE

Datový typ (TYPE) lze vybrat z následujících dostupných jednoduchých datových typů viz. Tabulka 5.3. Dále jsou uvedeny i sub-atributy a jejich významy. Složitější struktury je nutné zadat po jednotlivých proměnných a číst najednou pomocí COMMAND nebo použít ARRAY či COLLECTION.

Skupina	**Typ**	**Význam/Zápis/Sub-atributy**	
Aritmetické	INTEGER	celočíselná hodnota se znaménkem	
		INTEGER (size) { option option ... {value, description, help}, {value, description, help}, {value, description, help} }	
		size	velikost v bytech, nepovinný parametr, předdefinovaná hodnota je 1
		value, description, help	u některých hodnot (value) je možné specifikovat jejich význam krátkým komentářem v description a podrobným popisem kde help je odkaz na něj
		DISPLAY_FORMAT string; EDIT_FORMAT string;	nahrazuje option, formátování vstupu a výstupu, je použita syntaxe formátovacího řetězce ANSI C funkcí print a get
		MIN_VALUE expression; MAX_VALUE expression;	nahrazuje option, minimální a maximální hodnota, pokud je potřeba zadat více intervalů, použije se pro další intervaly číslovaných sub-atributů MIN_VALUE1, MAX_VALUE1, MIN_VALUE2…
		SCALING_FACTOR expression;	nahrazuje option, zobrazovaná hodnota je hodnota v zařízení násobená tímto koeficientem, používá se u zařízení zobrazujících velmi malá nebo značně velká čísla, když nejsou schopna takto extrémní hodnoty uložit
		DEFAULT_VALUE expression; INITIAL_VALUE expression;	nahrazuje option, oba dva atributy mají stejný význam – definují implicitní hodnotu proměnné, DEFAULT_VALUE může být specifikována jinou proměnnou, INITIAL pouze číselnou konstantou, INITIAL má větší váhu
	UNSIGNED_INTEGER	celočíselná hodnota bez znaménka	
		UNSIGNED_INTEGER (size) { option option ... } {value, description, help}, {value, description, help}, {value, description, help} }	
		size, DISPLAY_FORMAT, EDIT_FORMAT, MIN_VALUE, MAX_VALUE, SCALING_FACTOR, DEFAULT_VALUE, INITIAL_VALUE	viz. předchozí popis
	FLOAT	čtyřbytová s plovoucí řádovou čárkou	
		FLOAT { option option ... {value, description, help}, {value, description, help}, {value, description, help} }	
		DISPLAY_FORMAT, EDIT_FORMAT, MIN_VALUE, MAX_VALUE, SCALING_FACTOR, DEFAULT_VALUE, INITIAL_VALUE	viz. předchozí popis
	DOUBLE	číslo s plovoucí řádovou čárkou v dvojnásobné přesnosti než FLOAT	

		DOUBLE { option option ... {value, description, help}, {value, description, help}, {value, description, help} }	
		DISPLAY_FORMAT, EDIT_FORMAT, MIN_VALUE, MAX_VALUE, SCALING_FACTOR, DEFAULT_VALUE, INITIAL_VALUE	viz. předchozí popis
Výčtové	ENUMERATED	ENUMERATED (size) { option option ... {value, description, help}, {value, description, help}, {value, description, help} }	
		size, DEFAULT_VALUE, INITAL_VALUE	viz. předchozí popis
	BIT_ENUMERATED	BIT_ENUMERATED (size) { option option ... {value, description, help, function, status-class, actions}, {value, description, help, function, status-class, actions}, {value, description, help, function, status-class, actions} }	
		size, DEFAULT_VALUE, INITAL_VALUE	viz. předchozí popis
		function	stejný význam jako CLASS u proměnné
		status-class	
		action	specifikuje akci, která bude provedena, pokud bude tento bit nastaven
Indexové	INDEX	proměnná k indexování polí, při zobrazování je třeba vyhledat položku, na níž index ukazuje a neuvádět hodnotu indexu	
		INDEX (size) item-array;	
		item-array	uvádí do kterého pole je proměnná indexem
		size	viz. předchozí popis
Řetězcové	ASCII	typ pro ukládání posloupnosti znaků v ISO Latin-1	
		ASCII (size) { option option ... };	
		size, DEFAULT_VALUE, INITIAL_VALUE	viz. předchozí popis
	PASSWORD	typ datově a parametry stejný jako ASCII, liší se jen v zobrazování uložené hodnoty	
		PASSWORD (size) { option option ... };	
		size, DEFAULT_VALUE, INITIAL_VALUE	viz. předchozí popis
	BITSTRING	řetězec posloupnost bitů, interpretace není stanovena	
		BITSTRING (length) { option option ... };	
		lenght	specifikuje počet bitů zadaných v řetězci, tzn. počet bytů řetězce
		DEFAULT_VALUE, INITIAL_VALUE	viz. předchozí popis
Datum a čas	DATE_AND_TIME	posloupnost bytů vyjadřujících datum a čas podle specifikace Profibus DPV1	
		DATE_AND_TIME (size) { option option ... };	
			významy parametrů norma neuvádí
	TIME	posloupnost bytů vyjadřující datum dle Profibus DPV1	
		DATE (size) { option option ... };	
			významy parametrů norma neuvádí

Tabulka 5.3 – Datové typy definované v EDDL

Při pohledu do [3] je občas nutné místo do popisu s vysvětlením nahlédnout přímo do syntaktického zápisu, jelikož zejména v příkladech je spousta chyb a nepřesností.

V položce LABEL je uvedeno jméno proměnné, které bude zobrazovat hostitelské zařízení při manipulaci s touto položkou.
Dále následují nepovinné parametry (Constant unit, Handling, Help, Pre-/post-edit actions, Pre-/post-read actions, Pre-/post-write actions, Read timeout, Write timeout, Validity, Response

codes). Z nich uvádím pouze Handling, význam ostatních je možné opět nalézt v [3].

Důvod, proč jsem nepovinný parametr HANDLING vybral, je jeho zásadní význam pro popis proměnných v Profibus PA zařízení a přímá návaznost na COMMAND element jazyka EDDL. HANDLING totiž definuje, zda lze popisovanou proměnnou číst (READ) a zapisovat (WRITE). U Profibus PA zařízení pak přicházejí v úvahu 2 možné zápisy:

```
HANDLING READ;
```

nebo

```
HANDLING READ & WRITE;.
```

Po definici takto specifikované proměnné musí následovat již zmíněný element COMMAND. Jeho význam je podrobně rozebrán v 5.4.5, vysvětlující příklad prozatím uvádím pro snazší pochopení.

```
VARIABLE hodnota1{
        …
        HANDLEING READ;
}

COMMAND Cti_hodnota1{
        INDEX 10;
        OPERATION READ;
        TRANSACTION{
                REQUEST{
                }
                REPLY{
                        hodnota1;
                }
        }
}

VARIABLE hodnota2{
        …
        HANDLEING READ & WRITE;
}
```

```
COMMAND Cti_hodnota2{
        INDEX 11;
        OPERATION READ;
        TRANSACTION{
                REQUEST{
                }
                REPLY{
                        hodnota2;
                }
        }
}

COMMAND Pis_hodnota2{
        INDEX 11;
        OPERATION WRITE;
        TRANSACTION{
                REQUEST{
                        hodnota2;
                }
                REPLY{
                }
        }
}
```

5.4.4 Menu

Pomocí klíčového slova lze parametry, metody a další specifikace hierarchicky organizovat a hostitelská aplikace je v takto předpřipravené podobě může nabízet uživateli. Použití menu není povinné, ale pokud jej tvůrce EDD souboru připraví, značně usnadní orientaci.

Zápis vypadá následovně:

```
MENU name
{
        attribute attribute ...
}
```

kde atributy mohou být některé z následujících hodnot oddělené středníkem.

`Label` je povinný parametr zobrazovaný hostitelským zařízením zapsaný ve formátu `LABEL string;`

Seznam položek je uveden slovem `Items` a má tento zápis

```
ITEMS
{
        menu-item , menu-item , ...
}
```

kde `menu-item` může být `VARIABLE`, `METHOD` nebo další `MENU`.

Parametr `STYLE` a všechny následující jsou již nepovinné. `STYLE` může nabývat hodnot `DIALOG`, `WINDOW` nebo uživatelsky definované – zápis je `STYLE string;`

Metoda přístupu je popsána pomocí `ACCESS` parametru s možnými hodnotami `ONLINE`, `OFFLINE`, zápis je `ACCESS access-style;`

Poslední parametr je `Validity` vyjadřující platnost položky. Jedná se o klasickou booleovskou proměnnou a tudíž její hodnoty jsou `TRUE` nebo `FALSE`.

5.4.5 Command

Všechny základní elementy, které jsme doposud rozebírali, byly pouze definicí uspořádání datových položek. COMMAND však definuje, kde tyto položky získat a jak s nimi zacházet. Každý COMMAND musí mít své unikátní jméno, které je hned v prvním řádku zápisu.

```
COMMAND Name
{
        attribute; attribute; ...
}
```

Za atributy jsou pak dosazovány některé z následujících – `Block`, `Slot`, `Index`, `Operation`, `Connection`, `Module`, `Response Codes`, `Transaction`.

Při použití atributu `Block` je jasné, že se jedná o relativní adresování vzhledem k počátku bloku a dále se použije jen `INDEX number;`.

Pokud chceme adresovat absolutně – je to možné. Pak nespecifikujeme blok ale použijeme pro adresování dvojici `SLOT number;` a `INDEX number;`.

Pomocí parametru `OPERATION` je řečeno o jakou operaci se jedná. Dostupné hodnoty jsou `READ`, `WRITE`, `DATA_EXCHANGE` a `COMMAND` – tedy čtení, zápis, výměna dat a příkaz, kde příkaz znamená vyvolání příkazu uloženého v zařízení.

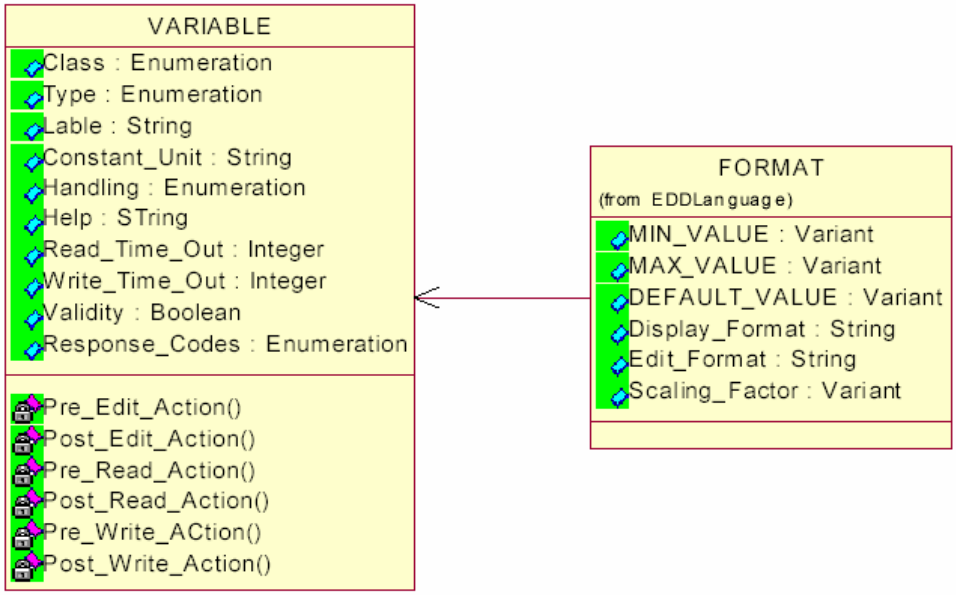
S jakými proměnnými budeme pracovat je popsáno pomocí `TRANSACTION` a použití je nejlépe patrné z následujícího příkladu.

```
...
OPERATION WRITE
TRANSACTION
{
    REQUEST
    {
        data-item , data-item , ...
    }
    REPLY
    {
    }
}
```

```
...
OPERATION READ;
TRANSACTION
{
    REQUEST
    {
    }
    REPLY
    {
        data-item , data-item , ...
    }
}
```

Pomocí RESPONSE_CODES je možné nadefinovat významy návratových hodnot po provedení této operace. Zápis vypadá takto.

```
RESPONSE_CODES
{
        value, type, description, help ;
        value, type, description, help ;
}
```



Obrázek 5.3 – Příklad definice VARIABLE pomocí UML

5.5 Příklad popisu v UML

Specifikace jazyka EDDL obsahuje i popis pomocí UML. Není možné jej však považovat za normativní zápis, je uveden spíše pro názornost a snazší pochopitelnost. Za pomoci UML je rozkresleno jen několik

základních a nejdůležitějších elementů jazyka EDDL. Na následujícím Obrázek 5.3 je uveden jako příklad popis elementu `VARIABLE`.

5.6 Zhodnocení

Popis zařízení pomocí EDD souboru je velmi mocný nástroj. Pro výrobce rozhodně jednodušší než tvorba vlastního programu. Možnosti, které nabízí, jsou zcela dostačující pro popis Profibus zařízení. Ve své komplexnosti je ale dosti svazující pro tvůrce konfiguračních nástrojů. Jejich práce se omezí pouze na implementaci čtení EDDL a vytvoření přístupu na sběrnici. Vše ostatní je již popsáno v EDD specifikaci zařízení.

Vytvoření dalších funkcí, které v EDD souboru popsány nejsou a výrobce je chce do programu zakomponovat, je poměrně komplikované. První možností je hybridní systém, část schopností bude vytvořena přímo v jeho kódu a zbytek bude realizován EDD serverem. Jedná se o způsob velmi nepřehledný.

Korektnější řešení by mělo rozšiřující funkce popsány také v EDDL a přistupovalo k nim přes EDD server. Tato varianta je natolik svazující, že je již předem odsouzena k zániku.

Za mnohem reálnější považuji postup, kdy aplikace postavená nad EDD soubory je nebude využívat do posledních detailů. Použije je pouze k načtení základní konfigurace bloků, proměnných a operací (stejně jsem postupoval i já) a zbylá inteligence programu bude vytvořena přímo v kódu.

Aby podobné nástroje začaly vznikat, musí se stát EDD soubor nedílnou součástí každého zařízení. Jinak se profesionální softwarové firmy do vývoje vůbec nepustí. I tak se domnívám, že jde o kvalitní nástroj k popisu Profibus zařízení s velkou budoucností.

6 Popis konfiguračního systému

6.1 Požadavky a cíle

Před začátkem práce na mém konfiguračním systému jsem si stanovil několik základních bodů a s ohledem na ně jsem směřoval celý další vývoj. Postupem času jsem některé postupy nahradil jinými, ale základní kostra zůstala nezměněna. Cílem tedy bylo vytvořit

- uživatelsky příjemný nástroj,
- s grafickým rozhraním,
- schopný konfigurovat zařízení na sběrnici Profibus PA,
- nástroj nezávislý na typu konfigurovaného zařízení
- nástroj nezávislý na výrobci zařízení,

- nástroj umožňující konfiguraci přes internet,
- nejlépe na platformě nezávislý.

První dva body jsou dnes samozřejmostí pro jakýkoliv program. Stačí umět používat některý z dostupných vývojových nástrojů a je téměř hotovo. Se zbylými body je to trochu komplikovanější.

6.1.1 Konfigurace na dálku

Jak tedy umožnit uživateli přístup ke konfiguračnímu nástroji na dálku, v tomto případě přes internet. Po zvážení různých variant jsem došel k závěru, že nejjednodušší bude navrhnout konfigurační systém s architekturou klient – server, tedy část systému spolupracující se sběrnicí Profibus PA bude provádět operace požadované připojenými klienty.

Architektura klient – server také částečně zjednodušuje problém přenositelnosti mezi platformami, neboť stačí vytvořit na platformě nezávislý klientský program. Internetové protokoly jsou dnes již tak rozšířené, že snad neexistuje platforma, na níž by nefungovaly.

6.1.2 Konfigurace nezávislá na typu a výrobci zařízení

Při vytváření nástroje nezávislého na typu a výrobci zařízení je většinou nutné otestovat a vyladit co největší množství dostupných modelů a tak alespoň s jistou pravděpodobností zajistit funkčnost pro libovolné zařízení. Rozsah mé diplomové práce a omezené prostředky tento způsob nedovolovaly a tak bylo nutné využít nějakého standardu, který by všechna zařízení dostatečně popsal a každé z nich jej splňovalo. Naštěstí profily zařízení Profibus PA popsané v kapitole 2.5 za takovýto standard považovat lze.

Standard však existuje jen v psané formě čitelné pro člověka, nikoliv pro počítač. Dalším krokem tedy bylo nalezení vhodného formátu informací o profilech, tak aby byl srozumitelný pro počítač. Původní návrh zvažoval možnost uložení profilů do databáze, z níž by následně informace čerpal jak server, tak klientské stanice. Tuto variantu jsem zavrhnul v okamžiku, kdy mi byl doporučen dokument popisující jazyk EDDL podrobně rozebíraný v kapitole 5. Ten je přímo určen k popisu libovolného zařízení Profibus.

6.2 Struktura systému

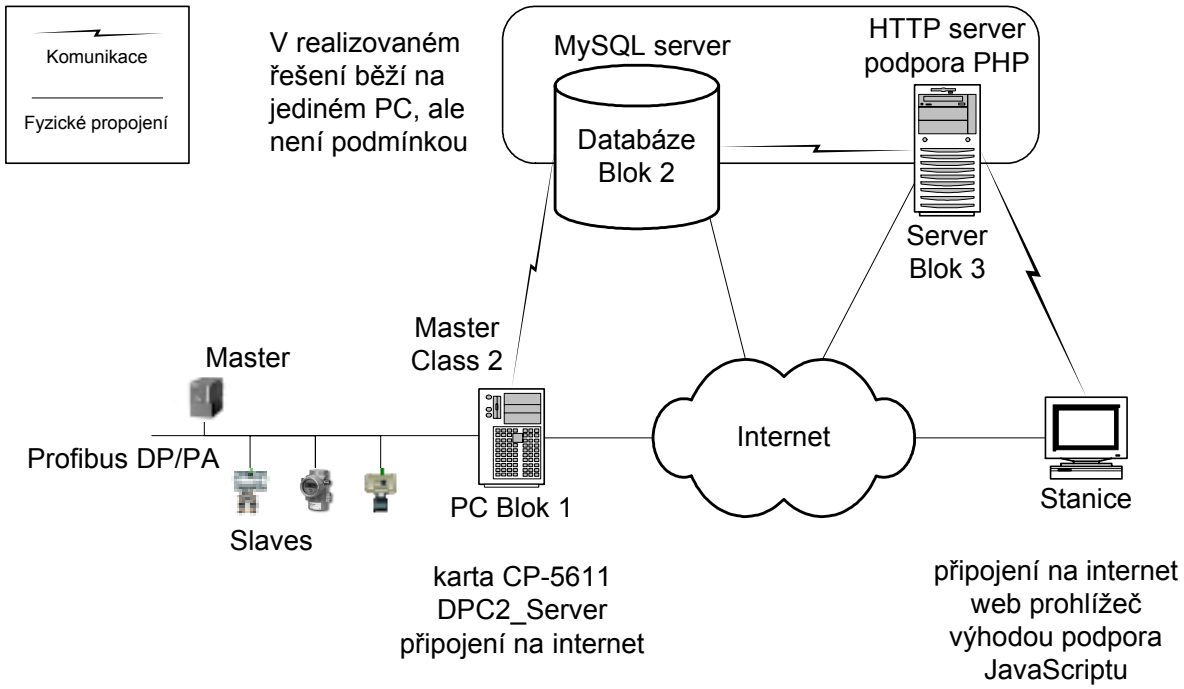
6.2.1 Nezávislé bloky

Celý konfigurační systém je rozdělen do tří nezávislých bloků – server, databázi a klienta. Jsou navrženy tak, aby každý z nich mohl běžet samostatně na zvláštním počítači. Takto strukturovaný systém je odolnější vůči chybám a snáze se spravuje a může využívat již existující HW vybavení přizpůsobené potřebám každého bloku.

První blok se stará o komunikaci se sběrnicí Profibus, zprostředkovává informace databázi a provádí operace požadované klienty. Tento blok je tvořen programem DPC2_Server a funguje jako server.

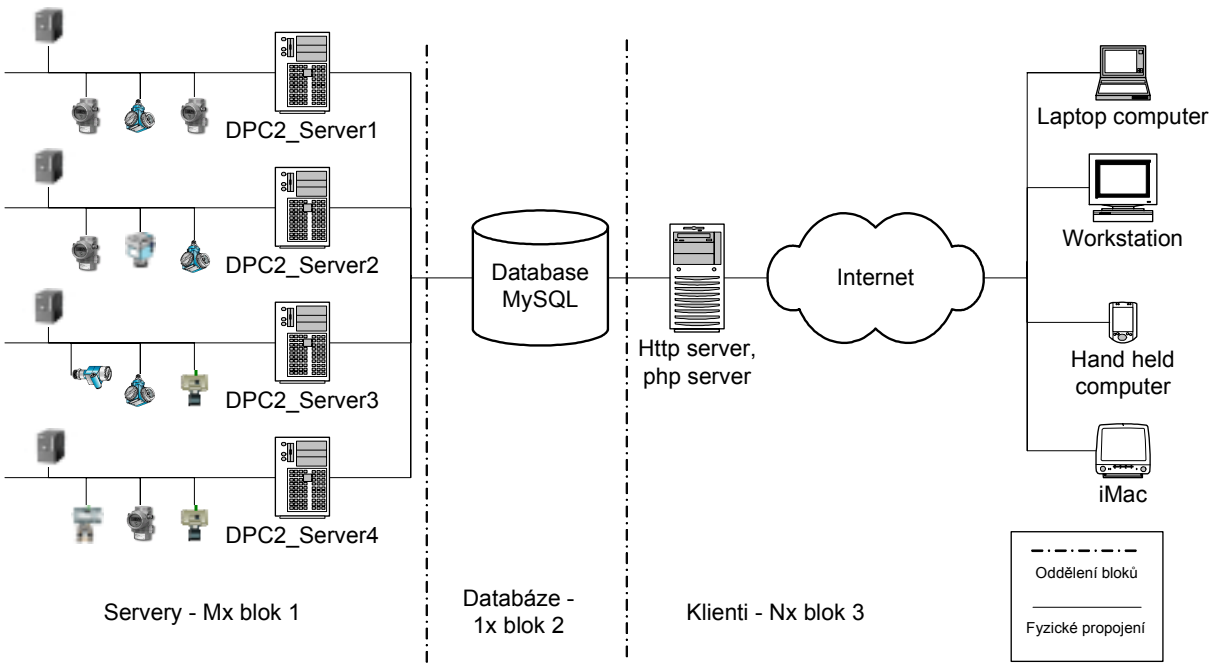
Druhý blok je databázová podpora klientů. Původní záměr byl používat tuto část pouze pro přenos dat a řídicí komunikace měla probíhat mezi serverem a klienty přímo. Nakonec byla i řídicí komunikace přesunuta do databáze a tím se tento druhý blok stal prostředníkem mezi klienty a serverem.

Třetí blok je webový klient se serverovými a klientskými skripty. Jeho serverová část běží shodou okolností na stejném počítači jako jeho databázová podpora, ale rozhodně to není nutnost. Klientská část se provádí přímo v internetovém prohlížeči uživatele. Klient je schopen prohlížet a pozměňovat údaje v zařízeních připojených ke sběrnici, kterou obsluhuje server. Více je patrné z Obrázek 6.1.



Obrázek 6.1 – Rozdělení systému do bloků

Takto vypadá nejjednodušší verze konfiguračního systému, ve které je každý blok obsažen právě jednou. Systém se však může skládat i z většího počtu bloků. Nejen že je schopen obsluhovat více klientů zároveň, k jedné podpůrné databázi je možno připojit i více serverových aplikací, a tak spravovat více technologií zároveň. Jejich počet závisí jen na rychlosti počítače na němž běží databázová podpora, aby byl schopen obsloužit všechny požadavky v rozumném čase.



Obrázek 6.2 – Modularita systému

6.2.2 Použité technologie

Technologie byly vybrány podle mnoha faktorů. Nejdůležitějšími byly potřebnost, dostupnost, použitelnost, perspektivnost a v neposlední řadě i moje alespoň minimální znalost a zkušenost.

Server běžící v bloku 1 využívá pro komunikaci se sběrnicí Profibus kartu CP-5611 a s ní spolupracující knihovnu DPC2Lib. Knihovnu dodává výrobce spolu kartou [15], ovládání je popsáno v [8]. Tato knihovna využívá možností rozšíření DPV1, které popisuje asynchronní komunikaci mezi DP Master a DP Slave zařízeními. Dále je v serveru použita knihovna LibMySQL pro komunikaci s blokem 2 – tedy databázovou podporou klientů. Volba knihoven LibMySQL a DPC2Lib byla rovněž ovlivněna zvoleným vývojovým prostředím, kterým je Microsoft Visual C++. MySQL server mi byl poskytnut pro blok 2 konfiguračního systému.

Klientská část systému je napsána jako webový klient. Její hlavní část je vytvořena v HTML, pro přístup k databázi jsem použil PHP skript (podpora PHP již na serveru běžela) a pro zvýšení uživatelského komfortu je klient doplněn o několik podpůrných funkcí v JavaScriptu.

Další zvolenou technologií je EDDL jazyk pro uložení informace o profilech zařízení na sběrnici Profibus PA. Jeho popis je možné najít v kapitole 5. Původně jsem chtěl informace uložit do databáze a používat je jak v klientské, tak serverové části systému. EDD popis se však pro ukládání do databáze příliš nehodí, a tak je použit spíše jako konfigurační soubor serverové části. Klientská část je pak napsána tak, že informace o profilech zařízení ke své práci nepotřebuje.

6.3 Server

6.3.1 Schopnosti

DPC2_Server je nejdůležitější část konfiguračního systému. Jak již bylo uvedeno výše, zajišťuje komunikaci mezi sběrnicí Profibus a databázovou podporou klientů. Zároveň lze přistupovat na sběrnici i přímo ze serveru pomocí vestavěného klienta. Jedná se vlastně o konečný automat, procházející následující stavy:

- načtení konfiguračních souborů
- připojení se ke kartě CP-5611
- vyhledání zařízení na sběrnici
- kompletní načtení proměnných v nalezených zařízeních
- pokud je dostupná databázová podpora, poskytnutí dat klientům
- pravidelné obnovování dat uložených v serveru
- pravidelné obnovování dat v databázi pro klienty
- provádění požadavků klientů
  - integrovaného klienta
  - vzdálených klientů prostřednictví databáze
- ukončení všech komunikačních kanálů
- odpojení se od karty.

Aby se mi podařilo zajistit uživatelský komfort, bylo nutné server rozdělit do několika vláken, která pracují souběžně. První vlákno je přiděleno základní aplikaci, aby ji bylo možné bez problému ovládat i při časově náročnější komunikaci s databází nebo se zařízeními na sběrnici Profibus PA.

Další vlákno je vytvořeno pro vyhledávání zařízení na sběrnici. Doba prohledání celé sběrnice, kde může být až 126 připojených zařízení, se pohybuje podle rychlosti sběrnice i v řádu několika minut. Po dokončení prohledávání sběrnice již vlákno není potřeba a je zrušeno.

Zbylá vlákna jsou přidělena instancím tříd, které obsluhují časově náročnější komunikace. Jsou to instance třídy `CDevice`, která je vytvořena pro každé zařízení připojené ke sběrnici Profibus. Druhou třídou vyžadující vlastní vlákno je `CMySQLComm`, tedy třída určená pro komunikaci s MySQL databází. Rozdělení vláken je dobře patrné ze schématu hierarchie tříd na Obrázek 6.3.

Spolu s rozdělením aplikace do více souběžně běžících vláken přichází další komplikace a tou je ošetření vícenásobných přístupů ke sdíleným proměnným. Problém je možné řešit několika běžně používanými způsoby jako jsou semafory, zamykání proměnných nebo komunikací prostřednictvím fronty zpráv. DPC2_Server využívá frontu zpráv a uzamykané proměnné, podle toho, která z variant byla v konkrétním případě výhodnější. Podrobnosti použití jsou uvedeny v odstavcích popisujících funkce nejdůležitějších tříd.

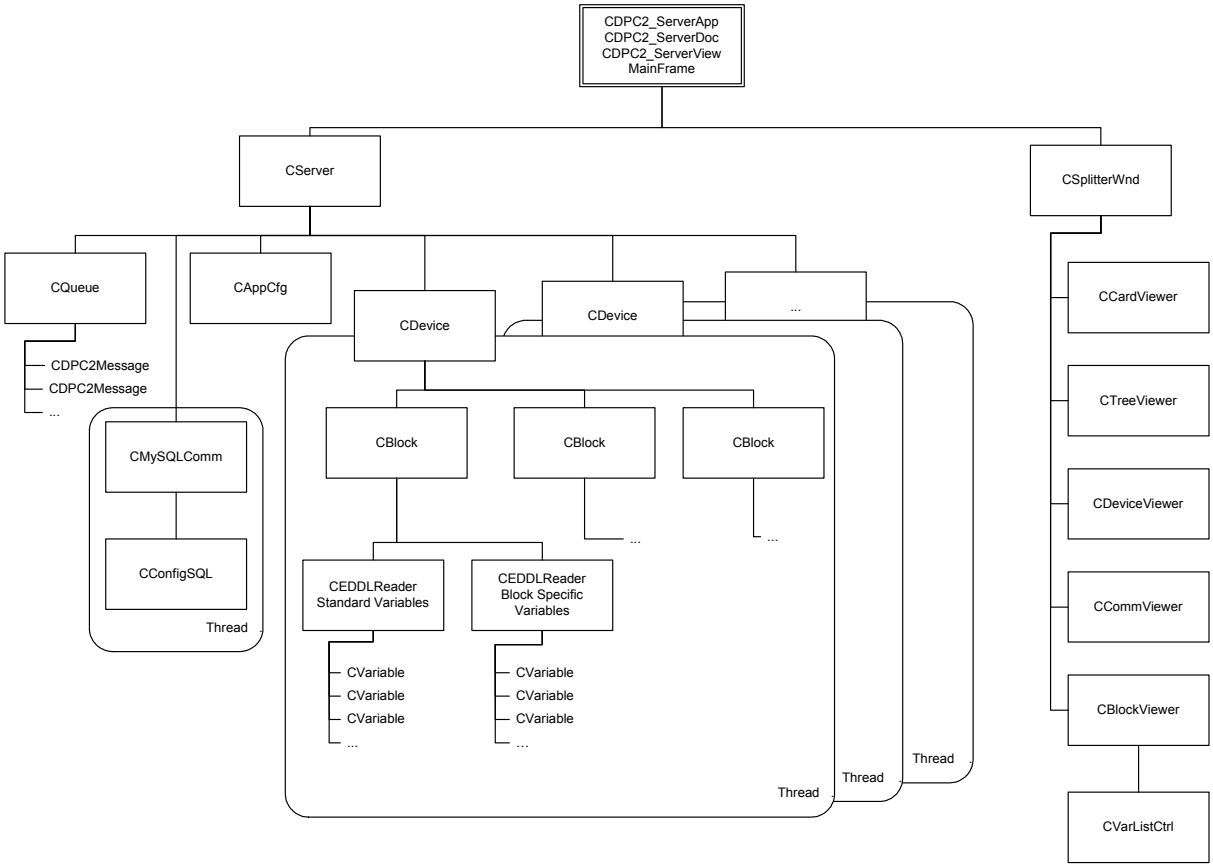
6.3.2 Vnitřní uspořádání

DPC2_Server se skládá z několika tříd, jejichž hierarchie je rozkreslena na Obrázek 6.3. Je možné ji rozdělit do tří hlavních částí.

Třídy `CDPC2_ServerApp`, `CDPC2_ServerDoc`, `CDPC2_ServerView` a `MainFrame` uvedeny v nejvyšší úrovni schématu se starají o chod aplikace pod MS Windows jako celku, obsluhu hlavního okna, přijímání zpráv, vykreslování oken, spolupráci s operačním systémem... Jejich struktura a způsob chování nejsou příliš zajímavé, protože jsou stejné pro všechny aplikace vyvinuté pomocí MFC[17] pod MS VC++. Jsou vytvářeny automaticky v rámci nového projektu.

Zajímavější je druhá vrstva hierarchie, kde se aplikace dělí na stěžejní část tvořenou třídou `CServer` a integrovaného klienta, který je tvořen třídou `CSplitterWnd` a třídami jí podřízenými.

Třída `CServer` má následující části: `CQueue` pro příjem a poskytování zpráv od karty CP-5611, `CAppCfg` pro nastavení volitelných parametrů, `CMySQLComm` pro komunikaci s databází MySQL a poslední částí je dynamicky vytvořený seznam instancí třídy `CDevice` pro každé zařízení připojené ke sběrnici Profibus.

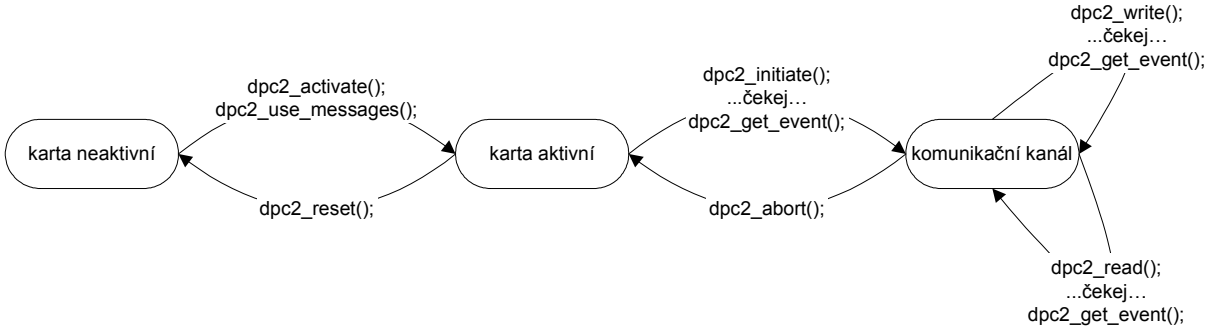


Obrázek 6.3 – Vnitřní uspořádání tříd DPC2_Serveru

[17] MFC – Microsoft Fundation Classes, sada knihoven pro MS Visual C++ určená k vytváření aplikací pod MS Windows, definuje okna, graf. komponenty, podpůrné třídy..., je nadstavbou Windows API funkcí

6.3.3 CServer

Když tedy shrnu vlastnosti třídy `CServer` uvedené v předchozích odstavcích, tato třída je vlastníkem všech instancí třídy `CDevice`, obsahuje jednu instanci třídy `CMySQLComm` pro komunikaci s MySQL serverem a dále obsahuje sadu funkcí pro zahájení komunikace s kartou CP-5611. Třídy `CDevice` a `CMySQLComm` budou popsány níže, zaměřme se na komunikační funkce `CServer`. Jak tedy komunikace s kartou funguje.



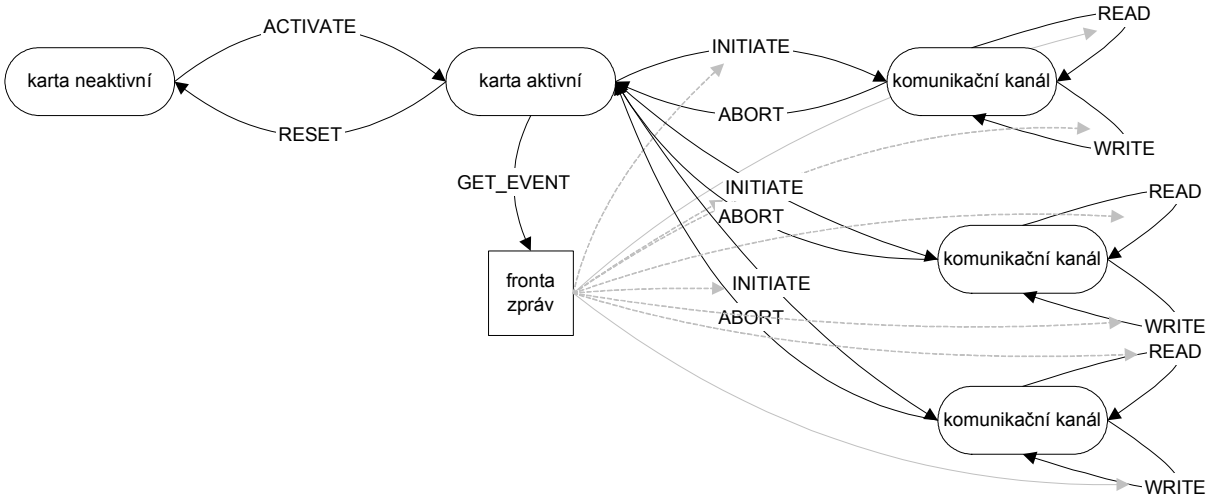
Obrázek 6.4 – Jednoduchá komunikace

Schéma jednoduché komunikace je rozkresleno na Obrázek 6.4. Nejprve je třeba pomocí funkce `dpc2_activate()` zahájit komunikaci s kartou. Jako parametr se funkci předává číslo logického jména v PG/PC interface (číslo `n` znamená logický název `CP_L2_n:/SCP`), typ komunikace (synchronní/asynchronní) a pointer na strukturu `dpc2_error` pro zapsání informace o případné chybě. Funkce vrátí ID karty, které je používáno v dalších funkcích (`dpc2_use_messages()`, `dpc2_reset()`). Protože hodláme komunikovat asynchronně, zaregistrujeme okno, které bude přijímat zprávy o dokončení jednotlivých operací karty, pomocí funkce `dpc2_use_messages()`.

Následuje otevření komunikačního kanálu s konkrétním zařízením funkcí `dpc2_initiate()`. Potřebné údaje pro spojení se vyplní do struktury `dpc2_initiate_rb`, která je jedním z parametrů funkce. Tentokrát již musíme čekat, až karta operaci dokončí a pošle nám zprávu. Tu vybereme pomocí `dpc2_get_event()`. Získáme doplněnou strukturu `dpc2_initiate_rb` např. o connection reference – parametr identifikující jednotlivé komunikační kanály.

Jakmile se nám podaří otevřít komunikační kanál, nic nebrání použití funkcí `dpc2_read()` pro čtení nebo `dpc2_write()` pro zápis. Opět fungují asynchronně, tzn. vyplníme patřičnou `dpc2_read_rb` nebo `dpc2_write_rb` strukturu, předáme ji funkci a čekáme, až nás karta vyzve k zavolání `dpc2_get_event()`.

Poslední dvě funkce slouží k ukončení komunikace. Kanál zavřeme funkcí `dpc2_abort()` a od karty se odpojíme funkcí `dpc2_reset()`.



Obrázek 6.5 – Složitější komunikace

V `CServeru` je celá komunikace rozložena do několika tříd a metod. Jedním z důvodů je i obsluha více kanálů zároveň. Proto bylo nutné vytvořit sdílenou frontu zpráv (`CQueue` podrobně rozebrána v 6.3.3.1) a do ní zprávy ukládat jako třídy `DPC2_Message`. Třída `CServer` nemá grafickou reprezentaci, není tudíž oknem a nemůže přijímat zprávy. Příjemcem je tedy `MainFrame` a pomocí metody `OnMessage()` `CServeru` vyvolá vybrání zprávy a její uložení pomocí funkce `dpc2_get_event()`. Ostatní komunikaci s kartou si již obsluhuje každá instance třídy `CDevice` zvlášť.

6.3.3.1 CQueue

Třída `CQueue` je vytvořena pro uchovávání zpráv od karty CP-5611. Protože si zprávy odtud vybírá nejenom třída `CServer`, ale i všechny instance třídy `CDevice`, je nutné data zabezpečit proti současnému přístupu více procesů zároveň. V tomto případě byla jako nejvýhodnější metoda použito uzamykání datových položek v kritických sekcích. Pokud jedna funkce vstoupí do kritické sekce, ostatní funkce vrací pouze hodnotu `FALSE` jako informaci o momentální nedostupnosti dat a je nutné je zavolat znovu. Jako příklad uvádím definici třídy a metodu `FindMessage()` pro vyhledání zprávy.

```
class CQueue
{
public:
        CDPC2_Message * GetMessage(unsigned short Order, unsigned short Event);
                //funkce předá pointer na zprávu podle pořadového čísla komunikace
                //a příslušné události
        BOOL RemMessage(unsigned short O);
                //funkce pro odstranění zpráv z fronty s pořadovým číslem O
        BOOL FindMessage(unsigned short Order, unsigned short Event);
                //vyhledá přítomnost požadované zprávy ve frontě, používá se zejména
                //při čekání na dokončení operace karty
        BOOL AddMessage(CDPC2_Message *);
                //přidání zprávy do fronty
        CQueue();
        ~CQueue();
                //vytvoření a zrušení fronty zpráv

private:
        CDPC2_Message * m_pFirst;           //spojový seznam zpráv
        BOOL m_bWorking;                    //zámek
        int iCount;                         //počet zpráv
```

```
};


BOOL CQueue::FindMessage(unsigned short Order, unsigned short Event)
{
	//vlastni hledani pozadovane zpravy
	if(m_bWorking) return FALSE;
	m_bWorking = TRUE;
	CDPC2_Message * Temp = m_pFirst;
	while((Temp != NULL)&&((Temp->usOrder!=Order)||(Temp->usEvent_type!=Event))){
		Temp = Temp->m_pNext;
	}

	if(Temp!=NULL){
		m_bWorking = FALSE;
		return TRUE;
	}
	else{
		m_bWorking = FALSE;
		return FALSE;
	}
}
```

Na metodě je vidět, jak probíhá uzamykání proměnných uvnitř třídy. Pokud již jiná metoda vstoupila do kritické sekce a nastavila si `m_bWorking` na `TRUE`, `FindMessage` nemůže dále pokračovat a končí stejnou návratovou hodnotou, jako kdyby hledanou zprávu nenašla. V opačném případě se sama uzamkne v kritické sekci a ve `while` cyklu prohledává zprávy. Jakmile dokončí hledání (ať úspěšně nebo neúspěšně), odemyká pomocí `m_bWorking` datové položky a vrací `TRUE` hodnotu pro úspěšné hledání či `FALSE` při neúspěchu.

6.3.3.2 CDevice

Třída `CDevice` se stará o komunikaci s jediným jí přiděleným zařízením Profibus PA. Je schopna s ním navázat a ukončit komunikaci, číst a zapisovat dle zadané adresy, rozpoznat adresářovou strukturu s počtem a pozicí bloků.

Třída `CServer` běží jako samostatné vlákno, je proto zděděna od MFC třídy `CWinThread`. Znamená to navíc po vytvoření instance této třídy konstruktorem ještě spuštění vlákna pomocí metody `CreateThread()`. Ostatní komunikace přicházející z tříd mimo vlákno probíhá pouze zasíláním zpráv, které se následně vyhodnocují v metodě `PreTranslateMessages()`.

Při vytváření instance třídy je nutné zadat dva povinné parametry. Jsou to adresa obsluhovaného zařízení a ukazatel na třídu `CServer` (hlavně kvůli přístupu k frontě zpráv `CQueue`).

Když `CServer` vyhledá všechna zařízení na sběrnici, spouští postupně ve všech instancích rozvinutí zařízení. Posílá jim proto zprávu `WM_EXPAND`, na kterou instance `CDevice` reagují zavoláním své metody `Expand()`. Během jejího provádění se přečte adresářová struktura (DO viz 2.5.3), nalezne se počet a pozice obsažených bloků a vytvoří pro ně patřičný počet instancí třídy `CBlock`. Každý `CBlock` se pak stará o data uložená v přiděleném bloku.

Po zbytek své existence `CDevice` jen přijímá požadavky na čtení, zápis, zobrazení či obnovení proměnných. Požadavky vyhodnocuje, ověřuje jejich korektnost a směřuje je na příslušnou instanci třídy `CBlock`. Pokud požadavek znamená vykreslení se do integrovaného klienta, vyvolá metodu `ShowDetails()`.

Jak probíhá vlastní komunikace se zařízením si ukážeme na příkladu zdrojového kódu metodu `ReadData()`. Jako vstupní parametry je potřeba uvést `Slot` a `Index` čtené adresy, připravit místo pro uložení čtených dat `csBuffer` a specifikovat jeho velikost `iBuffSize`.

```
int CDevice::ReadData(int iSlot, int iIndex, char * csBuffer, int iBuffSize)
{
	BOOL bRes = TRUE;
	if(!IsConnected()) bRes = Connect();
	if(!bRes){
		strcpy(csBuffer,"\0");
		return -1;
	}

	//priprav si strukturu
	struct dpc2_read_rb * Read_data = (dpc2_read_rb *)malloc(sizeof(struct dpc2_read_rb));
	//kontrola slot number
	if((iSlot<0)|||(iSlot>255)){
		m_pServer->m_pParent->MessageBox("Invalid Slot number!
<0,255>","Error",MB_ICONEXCLAMATION);
		free(Read_data);
		strcpy(csBuffer,"\0");
		return -1;
	}
	//kontola Index number
	if((iIndex<0)|||(iIndex>255)){
		m_pServer->m_pParent->MessageBox("Invalid Index number!
<0,255>","Error",MB_ICONEXCLAMATION);
		free(Read_data);
		strcpy(csBuffer,"\0");
		return -1;
	}
...
```

Zatím si metoda ověřila, že čtená adresa může existovat a nealokovala datovou strukturu `dpc2_read_rb` pro operaci čtení. Pokud je některý ze vstupních údajů zadán mimo povolený rozsah, objeví se chybové hlášení a metoda končí. Jinak v následující části datovou strukturu vyplní. `C_Ref` reprezentuje číslo komunikačního kanálu. Při zavolání funkce `dpc2_read` je třeba zadat unikátní číslo operace. To si metoda vyžádá od `CServeru` a uloží si jej do `myActionID`.

```
...
	//napln strukturu
	Read_data->C_Ref = m_iConRef;
	Read_data->Slot_Number = iSlot;
	Read_data->Index = iIndex;
	Read_data->Length_s = DPC2_DATA_LEN_S;
	Read_data->Data_s = (unsigned char *)malloc(sizeof(unsigned char)*DPC2_DATA_LEN_S);
	_strnset((char *)Read_data->Data_s,0,DPC2_DATA_LEN_S);
	//odesli zadost
	int myActionID = m_pServer->iActionID++;
	unsigned short result = dpc2_read(myActionID,Read_data);
	if(result!=DPC2_OK){
		m_pServer->ErrorEval(&(Read_data->error));
		if(Read_data->error.Error_Class==DPC2_ERROR_REQUEST_PARAM){
			if(Read_data->error.Error_Code==DPC2_ERROR_C_REF){
				m_iConRef=-1;
			}
		}
		strcpy(csBuffer,"\0");
		return -1;
	}
...
```

Pokud se požadavek čtení podařilo zadat a funkce odpověděla `DPC2_OK`, je opakovaně vyhledávána očekávaná zpráva o dokončení operace

ve frontě příchozích zpráv. Zhruba po 15 vteřinách neúspěšného čekání je nahlášena chyba. Jinak se provede vybrání zprávy, vyhodnocení datové části a promazání všech zpráv uložených v `CQueue` s číslem operace shodným s `myActionID`.

```
...
        //pockej na zpravu o provedeni
        int iCounter = 250;
        while(!(m_pServer->m_pMessageQueue->FindMessage(myActionID,DPC2_READ_EVENT)) &&
                iCounter>0){
                //cekej na zpravu o dokonceni teto operace
                Sleep(50);

                iCounter--;
                if(!m_pServer) iCounter = 0;
                if(!m_pServer->m_pMessageQueue) iCounter = 0;
        }

        if(iCounter==0){
                free(Read_data->Data_s);
                free(Read_data);
                TRACE("Bacha - Nedockal jsem se odpovedi na READ");
                strcpy(csBuffer,"\0");
                return -1;
                //nedockal jsem se odpovedi
        }
        dpc2_read_rb * Req_data = (m_pServer->m_pMessageQueue->GetMessage(myActionID,
                DPC2_READ_EVENT))->GetReadData();
        //vyhodnot zpravu
        if(Req_data==NULL) {
                strcpy(csBuffer,"\0");
                return -1;
        }

        int dat = Req_data->Length_s;
        if(dat > 0){
                if(dat>iBuffSize) Req_data->Data_s[iBuffSize] = 0;
                memcpy(csBuffer,Req_data->Data_s,iBuffSize);
        }
        else strcpy(csBuffer,"\0");
        //smaz vsechny zpravy s tim souvisejici

        while(!(m_pServer->m_pMessageQueue->RemMessage(myActionID))){
        }

        return min(dat,iBuffSize);
}
```

Poslední operací je zkopírování přečtených dat do dodaného bufferu a vrácení délky získaných dat jako návratové hodnoty metody.

6.3.3.3 CBlock

Každý blok v zařízení Profibus PA obsahuje základní proměnné a dále proměnné určené jeho typem. Proto první, co se provede po inicializaci třídy, je identifikace sama sebe. Přečte se `BlockObject` bloku ze zařízení a první čtyři položky (`Reserved, BlockObject, ParentClass, Class`) se použijí pro identifikaci. Porovnají se s daty uloženými v konfiguračním souboru *`block_ident.blo`*.

Konfigurační soubor pro určení bloku obsahuje pro každý dostupný typ bloku jeden řádek. Příklad popisu bloku i s významem jednotlivých sloupců je uveden v Tabulka 6.1.

První čtyři byty bloku	Maska pro výběr relevantních bitů	Typ bloku (fyzický, funkční, převodníkový)	Jméno varianty bloku	Soubor obsahující popis proměnných
0;1;128;0	0x00ff8000	Physical Block	Manufacturer specific	PhysicalBlock
0;2;1;1	0x00ffffff	Function Block	Input Analog	B_AnalogInputFB
0;2;3;1	0x00ffffff	Function Block	Control PID	B_PIDControlFB
0;3;1;2;	0x00ffffff	Transducer Block	Pressure Absolute	B_PressureTB
0;3;4;1;	0x00ffffff	Transducer Block	Level Hydrostatic	B_LevelTB

Tabulka 6.1 – Příklad dat v souboru *block_ident.blo* pro určení bloku

Správný blok je vyhledán tak, že se na data ze zařízení aplikuje maska a výsledek se porovná s daty ze souboru. Pokud se shodují, přečte se jméno bloku a název souboru s popisem datových položek bloku. V opačném případě se pokračuje na další řádek. Díky uspořádání podle jednotlivých typů bloků a masek je zaručeno nalezení správného bloku, pokud nejsou data zcela chybná. Podrobněji a včetně příkladu je vyhledání správného bloku popsáno v odstavci 6.3.5. Třída `CBlock` k tomu využívá metody `LoadBlockIdent()`, `ExtrackLine()`, `Identify()` a `ReadBlockObject()`.

`ReadBlockObject()` přečte `BlockObject` ze zařízení. `LoadBlockIdent()` nahraje ze souboru *`block_ident.blo`*.rozpoznávací údaje a pomocí `ExtractLine()` uloží řádek po řádku do pole `block_ident_data`. `Identify()` pak provede konečné prohledání a stanoví, o jaký typ bloku se jedná.

Po určení bloku se vytvoří dvě instance třídy `CEDDLReader`. Jedna pro standardní proměnné a druhá pro proměnné obsažené v právě rozpoznaném typ bloku. Následně se již obě instance starají o své proměnné a `CBlock` jim pouze předává řídicí požadavky. Pokud se jedná o požadavek vykreslení nebo výpisu do databáze, vyřizuje je sám.

6.3.3.4 CEDDLReader

Tato třída je učena pro uchovávání různorodých dat. Typy, názvy a počty datových položek se přečtou podle typu bloku z konfiguračního EDD souboru. Pro čtení EDD souboru je zapotřebí preprocesor a interpret jazyka EDDL (viz. kapitola 5). Není implementován kompletní interpret pro jeho značnou složitost, jsou čteny pouze využívané parametry a ostatní jsou přeskakovány. Při vytváření specializovaných EDD popisů je výhodné vycházet z používaných souborů a tak preprocesoru a interpretu příliš nekomplikovat proces čtení.

Aby bylo možné takto různorodá data ukládat, je použita speciální třída `CVariable`, která je schopna pracovat s daty různých typů. Je napsána na míru Profibus zařízením a tak i dostupné datové typy odpovídají datovým typům specifikovaným jazykem EDDL.

Pro inicializaci třídy jsou použity následující metody. `ExtractFile()` celý proces zaštiťuje. `LoadFile()` načte souboru daného jména, `EraseComments()` z načtených dat vymaže komentáře a příkazy pro preprocesor. Nakonec se použije `ExtractVar()` – metoda vybírající z předpřipraveného textu jednotlivé proměnné a načítající jejich parametry. Po úspěšném dokončení inicializace se nastaví `m_bSuccLoad`.

Zbylé metody `GetVar()`, `GetVars()` a `GetVarsCount()` umožňují přístup k uchovávaným proměnným. `GetVarsCount()` vrací jejich počet, `GetVar()` vrátí proměnnou na požadovaném indexu v poli, `GetVars()` pak předává pointer na celé pole položek typu `CVariable`.

6.3.3.5 CVariable

Třída `CVariable` je obdobou v MFC již existující třídy `COleVariant`. Slouží k ukládání proměnných různého typu a délky. Protože však byla vytvořena na míru této aplikaci, má několik dalších vylepšení a některé metody neimplementované. První výhodou je přesná podpora datových typů specifikovaných v EDDL. Tím se podstatně zjednoduší hlídání velikosti datových položek.

Další je ukládání výchozí hodnoty proměnné (Default), pokud je specifikována. U většiny položek v Profibus zařízeních, které lze zapisovat, existují výchozí hodnoty. Bylo by sice možné ukládat je do další instance, pokud jsou však najednou, jsou všechny operace jako zápis, ukládání a zobrazování značně jednodušší.

Posledním a poměrně důležitým vylepšením je ukládání dat ve správném pořadí. Zařízení Profibus totiž poskytují datové položky delší než jeden byte v přímém pořadí (Big Endian[18]) a nikoliv v pořadí obvyklém v počítačích s procesorem Intel (Little Endian[19]). Proto jsem musel implementovat obracení datových položek, aby je bylo možné použít ve standardních funkcích jazyka C a MFC. Obracení je třeba provádět u všech vícebytových položek kromě textových řetězců. Zcela specificky je potřeba pracovat s bitovými řetězci, u kterých navíc není normou definovaný způsob manipulace, a tudíž každé zařízení jej může používat jinak.

```
class CVariable
{
public:
      CVariable(const char * name);
      virtual ~CVariable();

      void SetSize(int iSize,int iType);  //nastaví typ a velikost položky
      void SetValue(void * pData);        //nastaví hodnotu položky, vstupem

      void * GetVal4Write(void * pBuffer, int iSize);
      //vrací hodnotu s pořadím bytů pro zařízení (BigEndian)

      void SetValueStr(char * pValue);
```

[18] Big Endian, způsob ukládání vícebitových dat v počítači, obvyklé u počítačů s procesory Motorola (Macintosh), byte s nejnižší hodnotou je uložen na nejnižší adrese

[19] Little Endian, způsob ukládání vícebitových dat v počítači, obvyklé u počítačů s procesory Intel, byte s nejvyšší hodnotou je uložen na nejnižší adrese

```
      void SetDefValueStr(char * pDefValue);
      //dostane string a podle typu proměnné uloží odpovídající hodnotu do
      //datové části třídy

      char * GetStrDefValue(char * csTo, int iRadix);
      char * GetStrValue(char * csTo, int iRadix);
      //vstupem je buffer pro řetězec a číselná soustava, na toto místo
      //pak uloží uloženou hodnotu jako text

      int m_iID;              //pokud je v databázi, její ID
      int m_iType;            //datový typ proměnné viz níže
      int m_iSize;            //počet bytů proměnné (m_pValue)
      char * m_sName;         //jméno proměnné
      int m_iOperations;      //Read = 1, Write = 2
      int m_iIndex;           //relativní pozice uvnitř bloku

private:
      BOOL m_bValue;          //hodnota dat existuje
      BOOL m_bDefValue;       //hodnota default dat existuje
      BOOL m_bReady;          //třída zinicializována
      void * m_pDefValue;     //default data
      void * m_pValue;        //data

      char * _GetStr(char * csTo, int iRadix, BOOL bValue, void * pValue);
      BOOL _SetStr(void *pTarget, BOOL * bValidity, char *pUnivValue);
};
```

Po vytvoření instance této třídy pomocí konstruktoru je nutné ji inicializovat pomocí `SetSize()` – nastaví se datový typ a počet bytů. Datové typy odpovídají specifikaci EDDL a využívají nadefinované konstanty:

```
//
//                EDDL DATA TYPES DEFINITION
//
//
#define           INTEGER                 1
#define           UNSIGNED_INTEGER        2
#define           FLOAT                   3
#define           DOUBLE                  4
#define           ENUMERATED              5
#define           BIT_ENUMERATED          6
#define           ASCII                   7
#define           PASSWORD                8
#define           BITSTRING               9
#define           DATE_AND_TIME           10
#define           TIME                    11
```

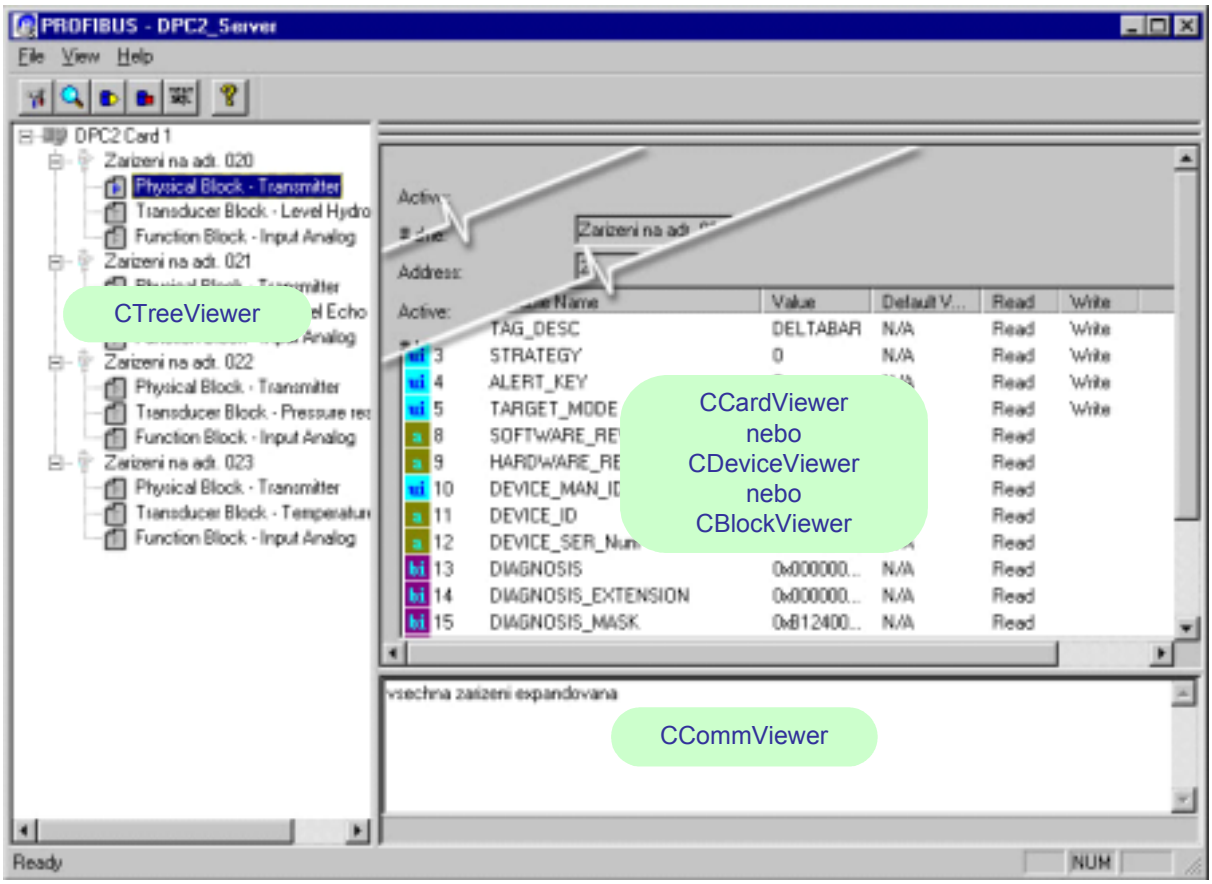
Po inicializaci se nastaví `m_bReady` na `TRUE` a je možné některou z mnoha metod zadat hodnotu nebo přednastavenou hodnoty. Je možné si vybrat mezi zadáváním přímo hodnotou nebo textovám řetězcem.

Protože metody `SetValueStr()` a `SetDefValueStr()` resp. `GetStrValue()` a `GetStrDefValue()` vlastně provádí stejné operace, pouze s jinými datovými místy, zavolají si vždy patřičnou metodu `_GetStr()` nebo `_SetStr()`. Podřízené metody pak převádí data na řetězce nebo řetězce na data a ukládají je na správná místa.

6.3.4 Integrovaný klient

Integrovaný klient původně vznikl pro účely ladění a testování chodu serverové části aplikace. Je schopen provádět stejné operace jako internetový klient, pouze k tomu nepotřebuje podpůrnou databázi. Skládá se z několika zobrazovacích oken popsaných na Obrázek 6.6. Po prohledání sběrnice jsou připojená zařízení společně s bloky v nich obsaženými zařazeny do stromu v `CTreeViewer`u pod implicitní ikonku karty. Následně se v pravé části aplikace mění typ zobrazovaných informací podle aktuálně vybrané položky stromu. Pokud je vybrána karta, zobrazuje se v okně typu CCardViewer stav karty, počet nalezených zařízení atd.

V okně typu `CDeviceViewer` jsou uloženy informace o vybraném zařízení. Poslední a zároveň nejzajímavější je okno typu `CBlockViewer`. Obsahuje totiž seznam standardních a specifických datových položek bloku. Dále jsou do okna vloženy ovládací prvky, s jejichž pomocí lze vyžádat opětovné načtení všech položek bloku, selektivní obnova jen jediné položky nebo přepsání hodnoty položky jinou hodnotou.



Obrázek 6.6 – Integrovaný klient

Pro zobrazování datových položek je vytvořena nová třída `CVarListCtrl`, poskytující dodatečné ovládací a informační funkce.

6.3.4.1 CVarListCtrl

`CVarListCtrl` je potomek standardní třídy obsažené v MFC s názvem `CListCtrl`. Původní třída je určená k zobrazování různorodých dat. Umožňuje několik pohledů obvyklých ve Windows jako velké ikony, malé ikony, detailní výpis atd. Je schopna k jednotlivým položkám přiřazovat několik sad ikon podle aktivního pohledu, řadit položky podle různých kritérií, průběžně položky zaměňovat, upravovat vyhledávat a rušit.

`CVarListCtrl` využívá pouze detailní výpis třídy `CListCtrl`. Oproti mateřské třídě má vylepšeny některé vykreslovací, vyhledávací a označovací funkce. Např. označení položky je možné provést po celé šířce okna, nikoliv jen ve sloupci s ikonou nebo relativní adresou položky.

Stěžejním vylepšením je implementace systému odesílání zpráv při změně aktuální položky. Nadřazené okno (v tomto případě `CBlockViewer`) dostává informaci o aktuálně vybrané položce. Může tak na základě přijatých zpráv měnit hodnoty a funkce ovládacích prvků pro čtení dat ze zařízení nebo jejich zápis. Bylo proto nutné redefinovat metody `OnKeyDown()`, a `OnLButtonDown()`, které reagují na vstupní zařízení – myš a klávesnice, kontrolují co se děje a v případě změny vybraného řádku odesílají zprávu `WM_LIST_CHANGED`. Zpráva sebou samozřejmě nese jako `WPARAM` i číslo vybraného řádku.

6.3.5 Konfigurovatelnost

Umožnit uživateli snadnou změnu konfigurace jsem si stanovil v základních vlastnostech programu. Co všechno tedy lze nastavit.

Již v kapitole 6.3.3.3 byl zmíněn konfigurační soubor pro určení bloků. Ten obsahuje kromě identifikace, jména a typu bloku i jméno souboru, kde se nachází podrobný popis bloku. Jméno souboru je uváděno pouze ve zkráceném tvaru, ke všem názvům podrobných popisů se ještě přidá *`_variables.ddl`*. Prozatím aplikace obsahuje jen popisy zařízení profilů uvedených v normě [2] a popsaných v odstavci 2.5. Pokud však výrobce nebo uživatel vytvoří EDD popis nějakému zařízení na míru, může EDD soubor přidat do adresáře k ostatním a dopsat další řádek do souboru *`block_ident.blo`*.

Aby vše fungovalo správně, je potřeba dodržet dvě podmínky. Za prvé blok musí být jedinečně identifikován prvními čtyřmi byty (`Reserved`, `BlockObject`, `ParentClass`, `Class`) a za druhé řádky souboru *`block_ident.blo`* musí být seřazeny podle typu bloku a hlavně podle složitosti masky od složitějších k obecnějším.

Pokud by nebyla dodržena druhá podmínka, použití obecnější masky na řádku s nižším pořadovým číslem by znamenalo vybrání tohoto řádku před testováním řádku se složitější maskou. Pro lepší pochopení uvádím příklad rozpoznávání bloku. Výpočty uvedené v Tabulka 6.2 jsou prováděny postupně shora dolů. Jakmile se objeví první shoda, hledání je zastaveno, v tomto případě na řádku 2. Pokud by byly řádky 2 a 3 prohozeny, blok by se chybně rozpoznal jako `PB – manufacturer specific` a řádek s popisem `PB – special` by již kontrolován nebyl. Obecnější maska tedy obsahuje více bitů nastavených na úroveň 0.

Přečteno ze souboru					
Block object přečtený ze zařízení	**Aplikovaná maska**	**Výsledek po použití masky**	**Porovnávaná hodnota**	**Rozhodnutí**	**Popis**
0x0001fd05	0x00ffff00	0x0001fd00	0x00010600	nesouhlasí	PB – lab. devices
0x0001fd05	0x00ffff00	0x0001fd00	0x0001fd00	souhlasí	PB – special
0x0001fd05	0x00ff8000	0x00018000	0x00018000	souhlasí	PB – man. specific

Tabulka 6.2 – Rozpoznání bloku

Dále je v programu možné nastavit, kde je umístěna databáze použitá v bloku 2 pro podporu klientů. Tlačítkem na nástrojové liště je možné provést změnu používané databáze a zároveň otestovat, zda je dosažitelná. Dále konfigurační okno obsahuje checkbox uložit nastavení. Pokud je vybrán, při uložení okna dojde k zapsání zadaných údajů do konfiguračního souboru, a tudíž budou použita při příštím spuštění programu.

Před spuštěním vyhledávání zařízení na sběrnici Profibus je také možné nastavit několik parametrů vyhledávání. Vyhledávání zařízení na poměrně pomalé sběrnici Profibus PA může zabrat poměrně dost času. Výrazně se dá zkrátit nastavením rozsahu portů hledaných zařízení. Pokud se např. nacházejí jen na adresách od 10 do 30, není třeba prohledávat adresy mimo tento rozsah. V tomto konkrétním případě klesne doba nalezení všech zařízení na cca 16%.

Všechny konfigurační soubory jsou umístěny v adresáři *Files*. Adresář je výhodné čas od času zálohovat, protože program si jeho obsah nedokáže při poškození sám nahradit. Obzvláště pokud jsou v některém ze souborů adresáře prováděny změny. Pro jistotu je originální podoba adresáře v komprimované podobě (*restore.zip*) v kořenovém adresáři aplikace.

6.4 Databáze

Databáze je druhým blokem konfiguračního systému. Zajišťuje přenos dat a komunikační propojení mezi serverem v bloku jedna a všemi připojenými klienty. Realizované řešení pracuje pouze s jediným serverem a několika málo klienty. Takto jednoduchý systém není schopen databázi vytížit natolik, aby bylo potřeba na její rychlost klást nároky. Pro složitější systém bude nutné na základě zátěžových testů nároky specifikovat. U serverové části se přetížení databáze příliš neprojeví díky samostatnému vláknu pro komunikaci s MySQL.

Podstatně nepříjemnější je pomalá odezva v klientském bloku. Po kliknutí na odkaz v prohlížeči se stránka nejprve smaže, poté čeká na dokončení PHP skriptu a až posléze se zobrazí požadované informace. Po celou dobu uživatel nemá tušení, co se děje. Na tento problém jsem narazil během vývoje klientské části systému při špatně formulovaném SQL dotazu.

6.4.1 Použitý databázový server

Jako databázový server mi byl poskytnut MySQL server produkovaný firmou MySQL AB. Server je v současné době hojně

používaným. Má však i svá omezení. S přihlédnutím k jeho omezením byla přizpůsobena celá databáze. Při návrhu byla zohledněna i možnost budoucí záměny databázového serveru. Tabulky jsou celkem jednoduché, nepoužívají zvláštní datové typy a nevyžadují provádění operací v transakcích (viz. níže).

6.4.2 Výhody a omezení MySQL

Současná značná oblíbenost MySQL serveru je způsobena zejména jeho šířením zdarma pod GNU General Public License (GPL). To znamená že aplikace, jejíž součástí je MySQL nebo jsou-li v ní použity ovladače pro MySQL (ODBC, JDBC, knihovny pro C), musí být nadále šířena jako GPL bezplatně a včetně kompletního zdrojového kódu. GPL aplikace také nemají žádné záruky výrobců MySQL a podporu v případě selhání serveru, knihoven nebo ovladačů.

Pokud si však uživatel nepřeje být omezován GPL, je pro něj dostupná i placená licence MySQL. Získá tak nejenom možnost šířit svůj produkt jako licenční ale zároveň je mu poskytována podpora ze strany MySQL AB.

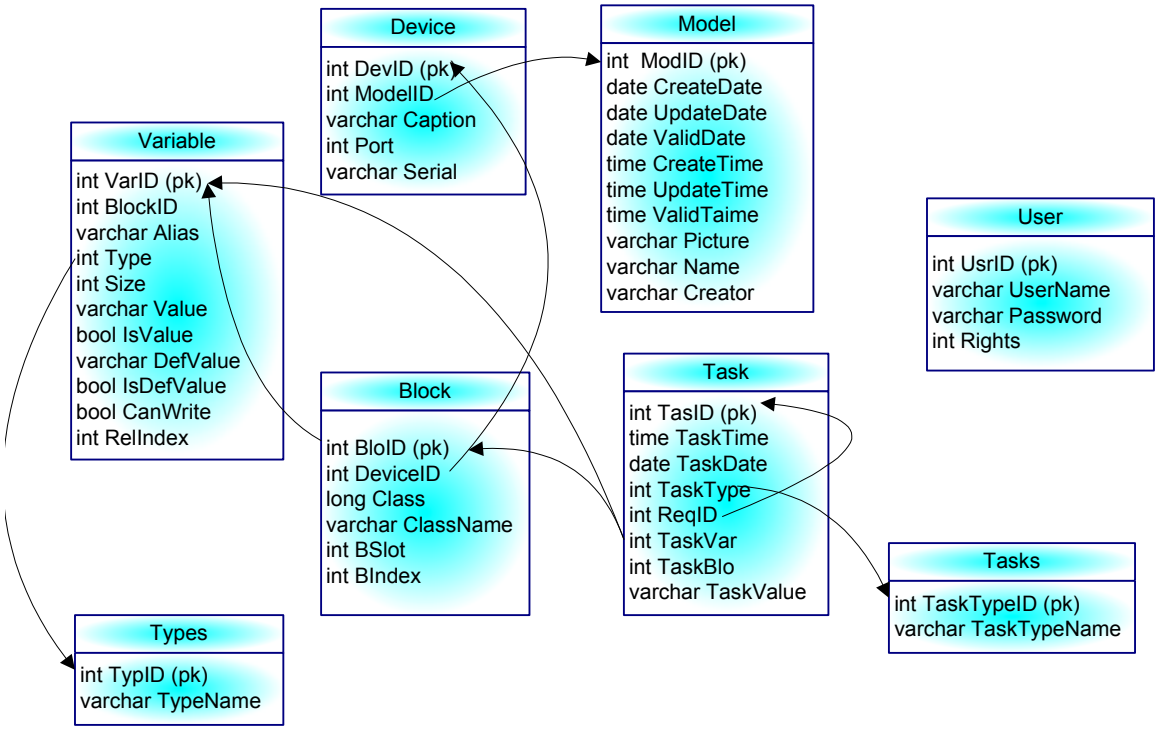
Mezi další výhody MySQL patří značná stabilita, spolehlivost. Rychlost MySQL je srovnatelná či lepší než většina konkurenčních a podstatně nákladnějších databázových systémů.

Nevýhodami je několik nepodporovaných operací. Patří mezi ně vnořené selekty nebo nepodporované transakce. Struktura databáze a používané dotazy byly vybrány tak, aby nepodporované operace nepotřebovaly.

6.4.3 Struktura databáze

Databáze obsahuje čtyři základní tabulky pro ukládání údajů z ovládané technologie. Jsou to `Model` pro popis jednotlivých technologií, `Device` pro ukládání informací o jednotlivých zařízeních, `Block` pro popis jejich bloků a `Variable`, do které se zapisují jednotlivé proměnné. Komunikaci mezi serverem a klienty zajišťuje tabulka `Task`.

Ostatní tabulky usnadňují funkci klientů (definice poznámek a vysvětlení – `Tasks`, `Types`) a starají se o správu uživatelů schopných k technologiím přistupovat (`User`). Relační model databáze je rozkreslen na Obrázek 6.7. V příloze v odstavci A je výpis vytvářecí sady SQL příkazů.



Obrázek 6.7 – Struktura databáze

Funkce většiny tabulek je zřejmá z modelu databáze, pouze tabulka `Task` je o něco složitější. Používá se ke komunikaci mezi serverem a klienty. Klient si vyžádá provedení operace zápisem následujícího řádku:

```
ID přidělí DB,
aktuální čas,
aktuální datum,
jedná se o Žádost -> tzn. 0,
NULL (jde o žádost - ta není s ničím svázána),
ID čtené/zapisované proměnné
NULL,
pokud zapisujeme – uvede se hodnota, jinak NULL.
```

Skutečný řádek pro čtení hodnoty pak vypadá takto:

```
NULL, '19:35:07', '2003-01-18', 0, NULL, 123, NULL, NULL
```

nebo pro zápis

```
NULL, '19:35:07', '2003-01-18', 0, NULL, 123, NULL, '1.123e+002'.
```

Server na tuto žádost odpovídá nejprve přijetím – začal jí zpracovávat – ve tvaru:

```
ID přidělí DB,
aktuální čas,
aktuální datum,
Přijetí -> tzn. 1,
Odkaz na ID žádosti,
ID čtené/zapisované proměnné ze žádosti
```

```
NULL,
NULL.
```

Skutečná podoba bude vypadat asi takto:

```
NULL, '19:35:50', '2003-01-18', 1, 1610, 123, NULL, NULL.
```

Pokud je žádost vyřízena bez problémů, `TaskType` bude 3. V případě chyby bude `TaskType` 2. Výsledné řádky pak vypadají:

```
chyba        NULL, '19:36:15', '2003-01-18', 2, 1610, 123, NULL, NULL
nebo úspěch  NULL, '19:36:15', '2003-01-18', 3, 1610, 123, NULL, NULL.
```

Takto je realizována veškerá komunikace mezi serverem a klienty.

Pro správu databáze je výhodné používat internetového klienta popsaného v dalším odstavci. Jednak má v sobě obsaženu podporu pro správu uživatelů. Další užitečnou pomůckou je správa modelů. Dokáže mazat staré a již nepoužívané modely z databáze včetně všech souvisejících položek v návazných tabulkách. Jeden model Profibus DP/PA demonstrátoru, s nímž jsem pracoval, je popsán zhruba 300 položkami. To obsahuje pouze 4 jednoduchá zařízení.

6.5 Internetový klient

6.5.1 Alternativy řešení

Co je vlastně internetový klient. V době značného rozšíření počítačových sítí a internetu bychom vše chtěli ovládat na dálku. Na jednom počítači připojeném k síti nám běží část systému schopná ovládat technologii (tzv. server), z druhého počítače na síti ji chceme řídit. K tomu potřebujeme klienta, pokud přistupuje k řídicí části přes internet, říkejme mu internetový klient. Může mít několik podob. Podle jeho složitosti, množství používaných dat a zakomponovaných znalostí hovoříme o dvou extrémních případech. Nazýváme je tlustý a tenký klient.

6.5.1.1 Tlustý klient

Tedy velmi robustní nástroj. Spolupráce se serverem bude přímá pomocí některé z vyšších objektových komunikací jako OLE[20], DDE[21], RPC[22]... Nástroj poměrně rychlý. Nevýhodou je náročnost na propustnost síťového připojení. To je způsobeno značnými komunikačními nároky. Při

[20] OLE – Object Linking and Embedding, technologie umožňující zabudovat cizí objekty do vlastní aplikace

[21] DDE – Dynamic Data Exchange, standardizovaná meziprocesní komunikace pod MS Windows

[22] RPC – Remote Procedure Call, vzdálené volání procedur fungující i v systémech běžících na různých platformách

srovnání komplikovanosti serveru a klienta půjde o stejně složité programy nebo klient bude složitější.

6.5.1.2 Tenký klient

Tenký klient je nástroj podstatně jednodušší. Bude vytvořen ve formě ne příliš komplikované aplikace nebo napsán v jazyce pouze interpretovaném (ASP[23], PHP[24], Java, VB…). Komunikovat bude patrně zprostředkovaně přes databázi nebo přímo pomocí některého z protokolů TCP/IP či jeho nadstaveb. Oproti tlustému klientovi bude téměř neinformovaným, tedy spíše vzdáleným ovládáním schopností serveru. Z toho vychází i porovnání složitosti. V tomto případě bude výrazně složitější server oproti jednoduchému klientovi.

Mezi těmito dvěma extrémními variantami existuje široká škála možných realizací. Pro každý řešený problém se hodí jiný kompromis mezi výhodami a nevýhodami předchozích variant. V poslední době je však snaha vytvářet systémy platformně nezávislé, a tudíž vychází výhodněji varianta bližší tenkému klientovi.

6.5.2 Zvolené řešení

Realizovaná forma klienta je tenký klient. Volba byla nejvíce ovlivněna cílem vytvořit platformně nezávislý systém. Protože jsem nechtěl vyvíjet několik klientů pro různé platformy, vytvořil jsem jediného webového klienta, který je schopen běžet na téměř jakékoliv platformě. Z dostupných technologií, které mi byly poskytnuty jsem vytvořil sadu html stránek doplněných skripty v jazycích PHP a JavaScript.

PHP skript jsem použil k získávání dat z podpůrné databáze a k jejich zobrazování, dále pak na propojení jednotlivých stránek a ukládání požadovaných operací do databáze.

JavaScript byl použit jen ke zvýšení komfortu klienta, provádí kontrolu provedení požadovaných operací v časových smyčkách a některé další podpůrné funkce.

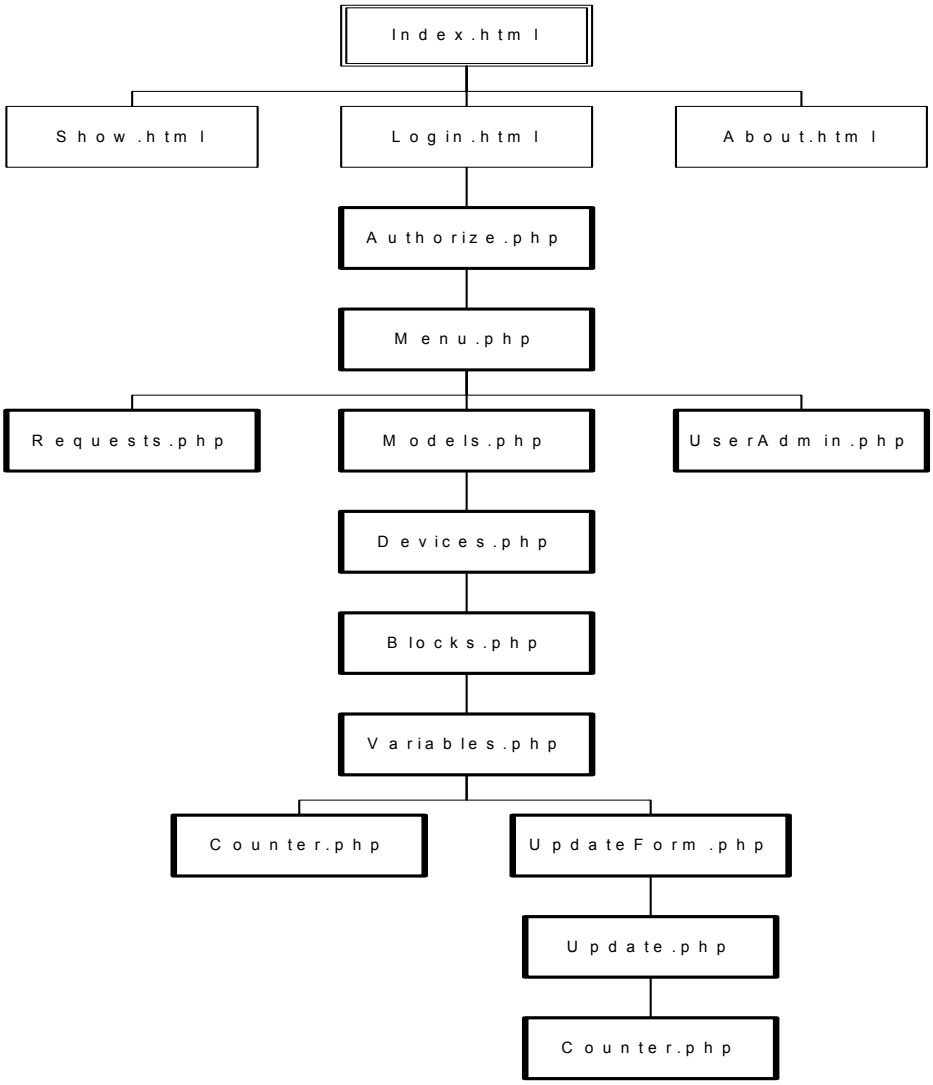
Také pro správu a řízení uživatelského přístupu je použito PHP skriptu.

6.5.3 Vlastnosti navrženého klienta

Na začátku se musí uživatel přihlásit. Klient rozlišuje tři druhy přístupových práv: Host, Uživatel, Správce. Podle práv přidělených použitému přihlašovacímu jménu jsou pak povoleny nebo zakázány některé funkce.

[23] ASP – Active Server Pages, skriptovací jazyk firmy Microsoft pro její internetové servery, jedná se o formu skriptu na straně serveru, po jejímž provedení zůstane jen klasická stránka ve formátu html, ta se následně zobrazí v uživatelově prohlížeči

[24] PHP, skriptovací jazyk, obdoba ASP, podporován např. web serverem Apache



Obrázek 6.8 – Struktura stránek internetového klienta

Nejnižší práva má Host. Ten smí pouze prohlížet modely dostupné v databázi a žádat o obnovení zobrazovaných hodnot. Žádosti lze provádět pouze u modelů, které jsou spravovány serverem. Zda je model spravován serverem nebo není se pozná podle data a času platnosti modelu. Při každém přístupu serveru do databáze je toto datum posunuto. Když server ztratí s databází spojení (server je vypnut, vypadne dlouhodobě komunikace…), platnost propadne a již nemá význam do databáze jakékoliv žádosti ukládat.

Standardní Uživatel smí navíc kromě obnovení požadovat i změnu údajů. Měnit lze pouze proměnné, u kterých to umožňuje profil zařízení. Opět zde funguje kontrola platnosti modelu a podle toho je ukládání žádostí povoleno nebo zakázáno.

Posledním typem uživatele je Správce. Jeho práva mu umožňují vše co smí Uživatel a navíc má dostupnou správu uživatelů, správu modelů a správu žádostí. Ve správě uživatelů lze vytvářet nové uživatele, měnit údaje o již existujících uživatelích a uživatele mazat. Správa modelů slouží ke korektní likvidaci již nepoužívaných modelů včetně všech souvisejících položek v ostatních tabulkách. Obdobně správa žádostí slouží k mazání žádostí patřících k již neobsluhovaným modelům.

Na Obrázek 6.8 je schématicky rozkreslena struktura internetového klienta. Stránky obsahující skripty jsou označeny tlustým rámečkem, stránky obsahující pouze html kód rámečkem tenkým. Čáry v obrázku neznamenají možné směry navigace, ale pouze hierarchii stránek. Všechny směry navigace včetně předávaných proměnných mezi jednotlivými stránkami je možné nalézt v podstatně podrobnějším obrázku v příloze C. V příloze jsou také stránky rozděleny na skupinu s grafickou reprezentací a skupinu zaměřenou pouze na provádění skriptů. Pokud stránka nemá grafickou reprezentaci, po provedení skriptu pokračuje automaticky v navigaci na další podřízenou stránku. Příklad skriptu i výsledné podoby jedné stránky je uveden v příloze D.

6.5.4 Umístění a přístupová práva

Realizovaného klienta spolu s několika modely je možné nalézt na internetových stránkách českého sdružení Profibus. Adresa zní `http://www.profibus.cz/pa/`. Pro přihlášení je potřeba znát jméno uživatele a heslo. Během vytváření databáze vznikají i tři základní uživatelé. Jména, hesla a práva jsou uvedena v následující Tabulka 6.3.

Dlouhé jméno	**Login**	**Heslo**	**Práva**
Administrator systemu	admin	paheslo	Správce
Navstevnik read-only	guest	guest	Host
Jan Novak	novakj	k12v24	Uživatel

Tabulka 6.3 - Základní uživatelé v klientské části

7 Závěr

V rámci řešení diplomové práce se mi podařilo dostát všech cílů, které jsem si vytyčil. V první řadě jsem ovládnul a naučil se používat kartu CP-5611 a podpůrnou knihovnu DPC2Lib pro synchronní a asynchronní komunikaci se zařízeními na sběrnici Profibus. Největší problémy jsem měl s dokumentací datových struktur a inicializací některých v této verzi knihovny zatím nepoužívaných proměnných. Při nesprávném nastavení jejich počátečních hodnot, přestože podle dokumentace nejsou požívány, program končil na neobsloužené výjimce uvnitř DLL knihovny. Její kód jsem byl nucen nakonec debugovat v asembleru, abych odhalil místo chyby.

Hlavní zkušenosti s kartou CP-5611 jsem získal při tvorbě programu DPC2_Analyser. Ten však byl schopen komunikovat jen s jedním zařízením v jeden okamžik. Dalším rozšířením proto bylo přepsání komunikace s kartou do více vláknového systému a ošetření přístupu ke zprávám od karty pomocí uzamykání proměnných.

Pro načítání popisu zařízení Profibus PA jsem realizoval jednoduchý interpreter EDD souborů – elektronického popisu zařízení. Kompletní překladač dle specifikace EDDL by byl díky své komplexnosti a obsáhlosti nad rámec diplomové práce.

Vytvořil jsem poměrně jednoduchého klienta, který je schopen konfigurovat zařízení Profibus PA přes internet. Jádro klienta používá pouze html a php. Díky tomu běží na široké škále platforem a nemá žádné nároky na konfiguraci nebo instalaci.

Při dalším vývoji by bylo dobré zdokonalit EDDL interpretr, aby se byl schopen vypořádat i se složitějšími a třeba i ne zcela korektními EDD popisy.

Současně s rozšířením schopností EDDL interpretru by mohly být doplněny EDD soubory o textové popisy výčtových parametrů. Na to navazuje i rozšíření klientů o schopnost textové významy hodnot zobrazovat.

Serverová část by šla doplnit o spoustu užitečných schopností, jako je periodického ukládání měřených veličin do historických tabulek (např. pomocí OPC Historical Data Server), jejich export a vytváření grafů.

Klientská část (zejména pak integrovaný klient) by mohla být schopna ukládat kompletní konfigurace jednotlivých zařízení do souboru či databáze. Následně by je načítal a konfigurovala celé zařízení najednou.

Závěrem bych chtěl poděkovat Ing. Pavlu Burgetovi za podporu a věcné připomínky během řešení diplomové práce. Janu Kelbelovi za uvedení do problematiky EDD popisu zařízení a poskytnutí EDD konfiguračních souborů pro profily zařízení Profibus PA. Katedře řídicí techniky za poskytnuté vybavení. Nakonec pak všem přátelům, kteří mi pomáhali při ovládnutí MS VC++ a MFC.

8 Použitá literatura

[1] PROFIBUS - DP Extensions to EN 50170 (DPV1), version 2.0, PNO, duben 1998
[2] PROFIBUS - PA Profile for Process Control Devices, version 3.0, PNO, říjen 1999
[3] Specific. for PROFIBUS Device Description and Device Integration, Volume 2: EDDL v 1.1, PNO, leden 2001
[4] Ondřej Dolejš, Komunikace pro průmyslovou automatizaci, Diplomová práce, ČVUT v Praze, 2000
[5] Miroslav Ličko, Průmyslová sběrnice Profibus, Diplomová práce, ČVUT v Praze, 2000
[6] Petr Smolík, Průmyslová sběrnice Profibus DP Extended, Diplomová práce, ČVUT v Praze, leden 2001
[7] Libor Švehla, Průmyslové komunikace - realizace DP-Slave, ČVUT v Praze, 2000
[8] Thomas Wacker, SIMATIC NET DPC2 Programming Interface

Internetové adresy

[9] http://www.profibus.cz
[10] http://www.profibus.com
[11] http://www.mysql.com
[12] http://www.php.com
[13] http://www.endress.com
[14] http://www.ifak.fhg.de
[15] http://www.siemens.com

9 Přílohy

A Vytvoření podpůrné databáze

B Ovládání serveru a integrovaného klienta

C Ovládání web klienta

D Příklad php a html kódu – Models.php

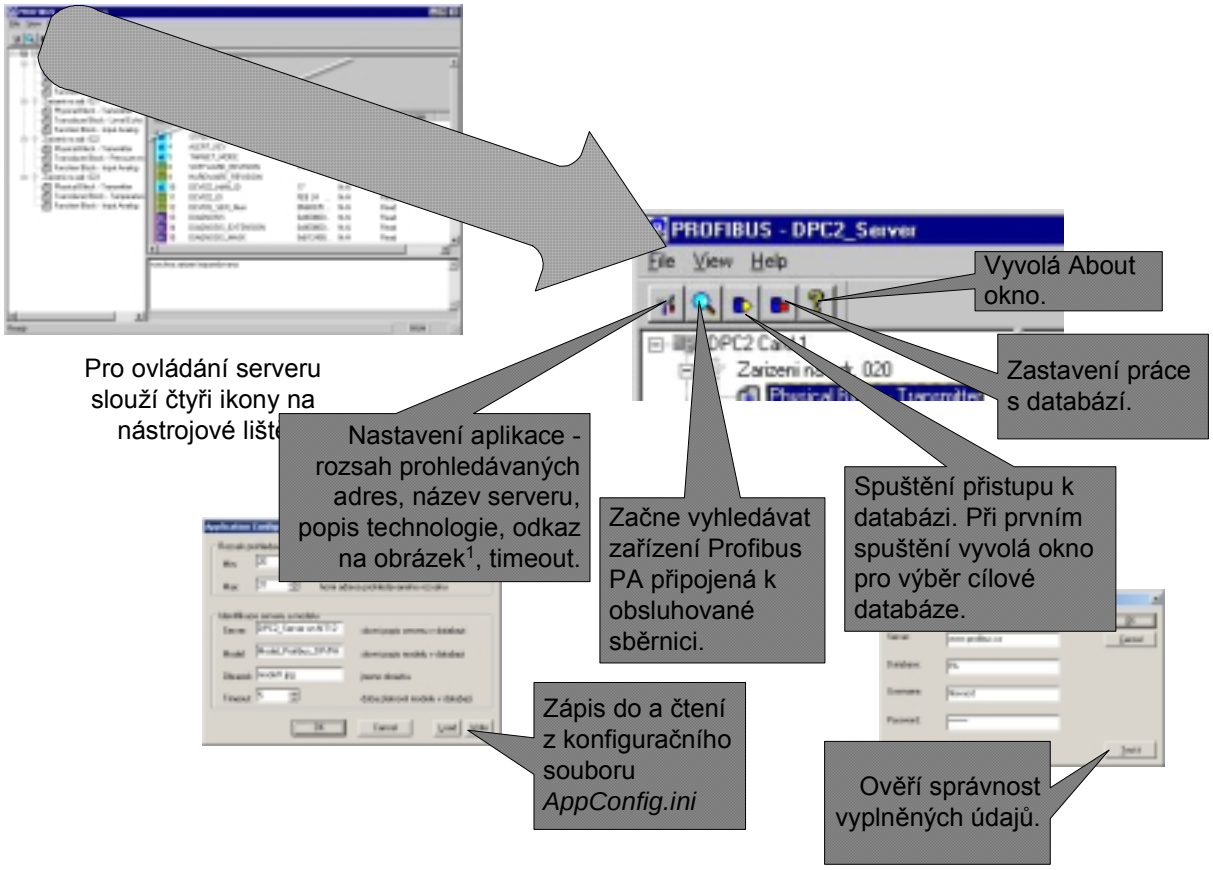
Příloha A – Vytvoření podpůrné databáze

```
# Connection: ProfibusCZ
# Host: www.profibus.cz
# Saved: 2003-01-10 13:25:13
#
create database PA;
use PA;

create table Model (
  ModID int unsigned not null auto_increment primary key,
  CreateDate date not null,
  UpdateDate date not null,
  ValidDate date not null,
  CreateTime time not null,
  UpdateTime time not null,
  ValidTime time not null,
  Picture varchar(100),
  Name varchar (20) not null,
  Creator varchar(20)
);

create table Device(
  DevID int unsigned not null auto_increment primary key,
  ModelID int unsigned not null,
  Caption varchar(30),
  Port int unsigned not null,
  Serial varchar(20)
);

create table Block(
  BloID int unsigned not null auto_increment primary key,
  DeviceID int unsigned not null,
  Class bigint(4) not null,
  ClassName varchar(50),
  BSlot int not null,
  BIndex int not null
);

create table Variable(
  VarID int unsigned not null auto_increment primary key,
  BlockID int unsigned not null,
  Alias varchar(50) not null,
  Type int not null,
  Size int not null,
  Value varchar(50),
  IsValue bool not null,
  DefValue varchar(50),
  IsDefValue bool not null,
  CanWrite bool not null,
  RelIndex int not null
);

create table Task(
  TasID int unsigned not null auto_increment primary key,
  TaskTime time not null,
  TaskDate date not null,
  TaskType int not null,
  ReqID int unsigned,
  TaskVar int unsigned,
  TaskBlo int unsigned,
  TaskValue varchar(50)
);

create table UserDetails(
  UsrID int unsigned not null auto_increment primary key,
  UserName varchar(20) not null,
  UserPass varchar(20) not null,
  UserRights int not null,
  UserNameLong varchar(100)
);
```

```sql
INSERT INTO Userdetails
VALUES( NULL,'admin','paheslo',100,'Administrator systemu' );
INSERT INTO Userdetails
VALUES( NULL,'guest','guest',1,'Navstevnik read-only' );
INSERT INTO Userdetails
VALUES( NULL,'novakj','k12v24',10,'Jan Novak' );

create table Types(
 TypID int unsigned not null auto_increment primary key,
 TypeName varchar(20) not null
);

INSERT INTO Types VALUES (1,"INTEGER");
INSERT INTO Types VALUES (2,"UNSIGNED_INTEGER");
INSERT INTO Types VALUES (3,"FLOAT");
INSERT INTO Types VALUES (4,"DOUBLE");
INSERT INTO Types VALUES (5,"ENUMERATED");
INSERT INTO Types VALUES (6,"BIT_ENUMERATED");
INSERT INTO Types VALUES (7,"ASCII");
INSERT INTO Types VALUES (8,"PASSWORD");
INSERT INTO Types VALUES (9,"BITSTRING");
INSERT INTO Types VALUES (10,"DATE_AND_TIME");
INSERT INTO Types VALUES (11,"TIME");

create table Tasks(
 TTyID int unsigned not null auto_increment primary key,
 TaskTypeName varchar(15) not null
);

INSERT INTO Tasks VALUES (0,"Zadost");
INSERT INTO Tasks VALUES (1,"Prijat");
INSERT INTO Tasks VALUES (2,"Odmitnut");
INSERT INTO Tasks VALUES (3,"Zpracovan");
```

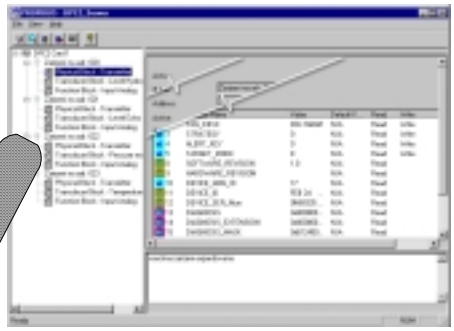
Příloha B – Ovládání serveru a integrovaného klienta

Server

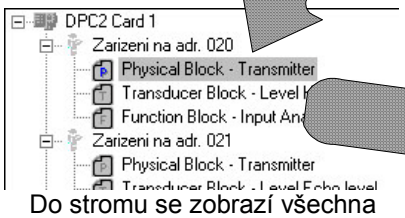


[1] Obrázky se zobrazují v klientovi při výpisu dostupných zařízení modelu. Jsou uloženy v adresáři models

Integrovaný klient

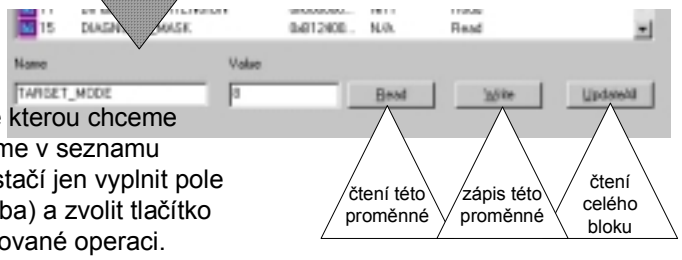


Celý klient se skláda ze tří oken - stromu vlevo, podrobností vpravo a výpisu komunikace ve spodní části.



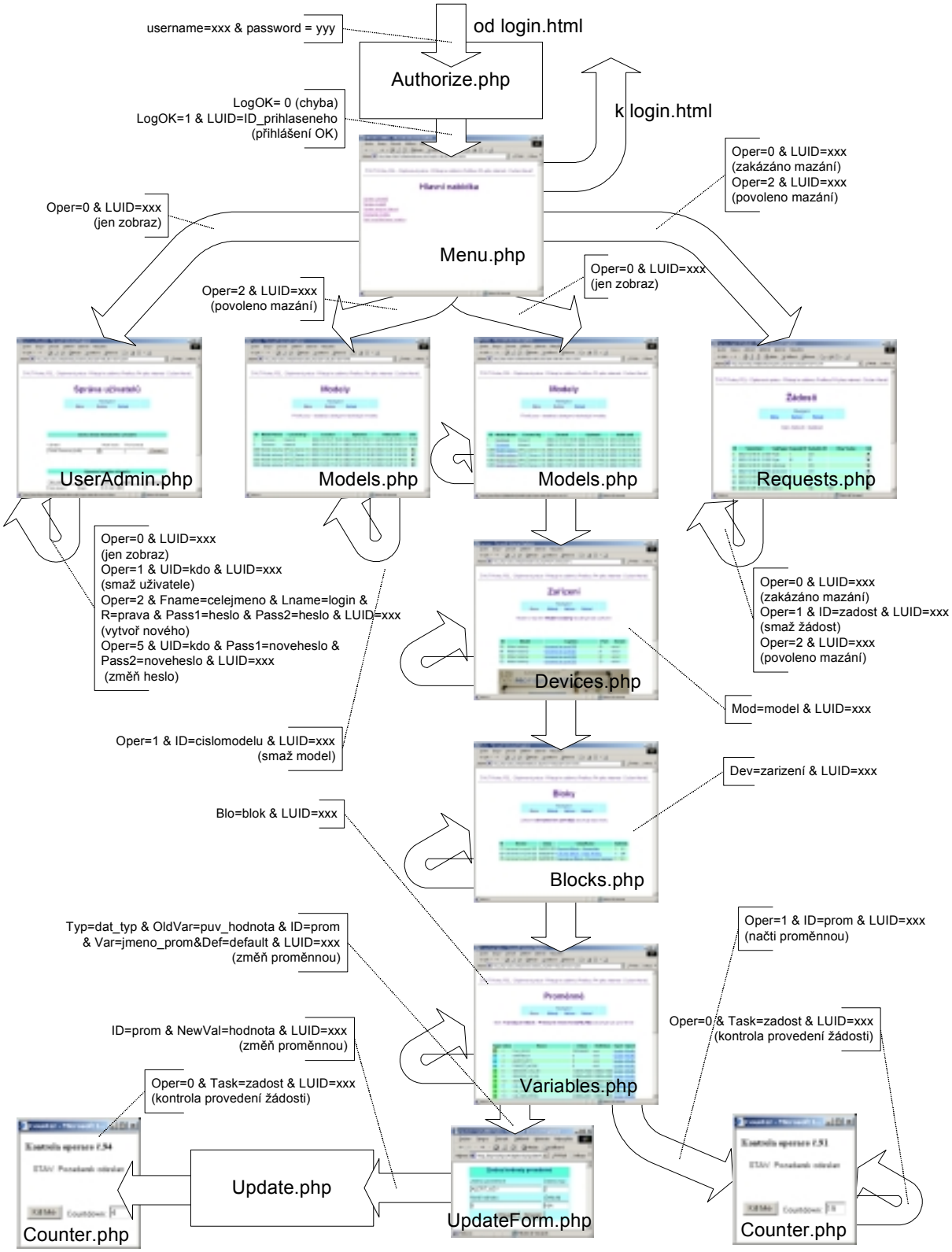
Do stromu se zobrazí všechna nalezená zařízení včetně bloků v nich obsažených. Pokud kliknutím myši vybereme některou položku ze stromu, v pravé části klinta uvidíme podrobné informace.

Pokud vybereme ve stromu blok, můžeme u zobrazených proměnných bloku žádat o opětovné načtení nebo pokud je to možné i o zápis nové hodnoty



Výběr proměnné, se kterou chceme pracovat, provedeme v seznamu proměnných výše. Pak stačí jen vyplnit pole VALUE (pokud je potřeba) a zvolit tlačítko odpovídající požadované operaci.

Příloha C – Ovládání web klienta



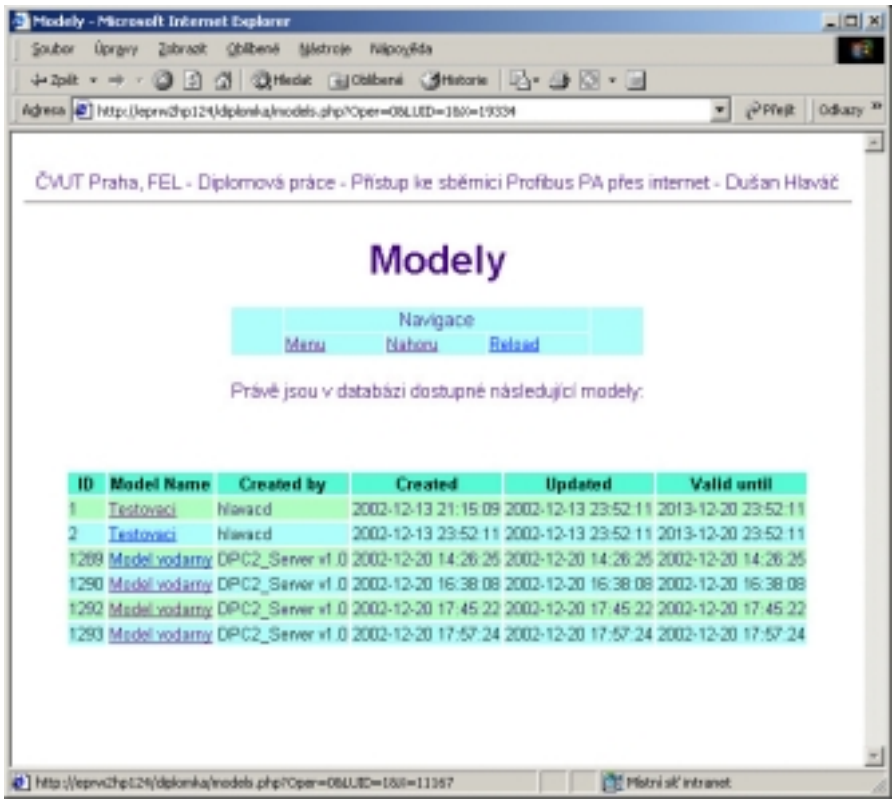
- U většiny oken se lze pohybovat ještě o úroveň výš, do menu a k výpisu modelů. V rámci zachování přehlednosti byly tyto směry v nákresu vypuštěny.
- Pokud popis nemíří k šipce, ale k celému oknu, platí pro všechny šipky mířící do okna.
- Okna bez obrázků nemají grafickou reprezentaci, po provedení php skriptu navigace rovnou přechází na následující okno.

Příloha D – Příklad php a html kódu – Models.php

```
            <html>

<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
<link REL="StyleSheet"  type="text/css" href="style_a.css">
<title>Modely</title>
</head>
<?php
        function connect(){
                $file = fopen("config.tzc","r");
        while(fscanf($file,"%s\t%s\t%s",$key,$value,$comment)){
                $data[$key] = $value;
        }
        fclose($file);
        // spojení s databází
        $conn = MySQL_Connect($data['dbserv'],$data['dbuser'],$data['dbpass']);
        if(!$conn){
                //echo mysql_error();
            return NULL;
        }
        mysql_select_db($data['dbname'])
                        or die("Nelze vybrat databázi");
                return $conn;
        }

        $hlaska = "";
        //-----------------------------------------------
        // spojení s databází
        $conn = connect();
        if(!$conn){
                $hlaska = "$hlaska, <p>Chyba spojeni s databazi</p>";
                exit;
        }

        //-------------------------------------------------------
        //standardni zjisteni prav prihlaseneho uzivatele
        $luid = $_GET['LUID'];

        $query = sprintf("SELECT UsrID, UserRights FROM UserDetails WHERE UsrID=%s",$luid);
        //print("<p>$query</p>");
        $result = mysql_query($query);
        if (!$result) {
                echo mysql_error();
                exit;
        }
        $rights=0;
        if($line = mysql_fetch_array($result, MYSQL_NUM)) {
                $rights=$line[1];
        }
?>

<body>
 <table border="0" cellpadding="0" cellspacing="0" width="100%">
    <tr>
      <td width="100%"> </td>
    </tr>
    <tr>
      <td width="100%">
      <p align="center">&#268;VUT Praha, FEL - Diplomová práce - P&#345;ístup ke sb&#283;rnici
      Profibus PA p&#345;es internet - Dušan Hlavá&#269;</td>
    </tr>
    <tr>
      <td width="100%"><hr></td>
    </tr>
  </table>

  <h1>Modely</h1>

  <!----- Odkazovaci tabulka -------------------------->

  <p><table border="0" cellspacing="1" width="50%">
```

```
  <tr>
    <td class="1" width="10%" rowspan="2"> </td>
    <td class="1" width="80%" colspan="3"><p>Navigace</p></td>
    <td class="1" width="10%" rowspan="2"> </td>
  </tr>
  <tr>
    <td class="1" width="20%"><a href="menu.php?LogOK=1&LUID=<? echo $luid ?>">Menu</a></td>
    <td class="1" width="20%"><a href="menu.php?LogOK=1&LUID=<? echo $luid ?>">Nahoru</a></td>
    <td class="1" width="20%"><a href="models.php?Oper=<? echo $_GET['Oper'] ?>&LUID=<? echo
$luid ?>&X=<? echo rand(); ?>">Reload</a></td>

  </tr>
</table></p>

   <p>Právě jsou v databázi dostupné následující modely:</p>

   <p> </p>
           <?
                   $delete = 0;//zakazano mazani modelu
                   if($_GET['Oper']==0){
                           //echo "Jen zobraz";
                   }
                   if(($_GET['Oper']==1)&&($rights==100)){
                           $delete = 1;
                           //delete all items from model with ID=$_GET['ID'];
                           //nalezeni vsech podrizenych polozek k modelu
                           $query = sprintf("select DevID from Device where
ModelID=%s",$_GET['ID']);
                           //print "\t<p class=\"1\">$query</p>\n";
                           $result = mysql_query($query);
                           while ($line = mysql_fetch_array($result, MYSQL_NUM)) {
                                   //nalezeni vsech podrizenych polozek k zarizeni
                                   $query1 = sprintf("select BloID from Block where
DeviceID=%s",$line[0]);
                                   //print "\t\t<p class=\"2\">$query1</p>\n";
                                   $result1 = mysql_query($query1);
                                   //hledani dal
                                   while ($line1=mysql_fetch_array($result1,MYSQL_NUM)){
                                           //smazani vsech podrizenych polozek k bloku
                                           $query2 = sprintf("delete from Variable where
BlockID=%s",$line1[0]);
                                           //print "\t\t\t<p class=\"3\">$query2</p>\n";
                                           $result2 = mysql_query($query2);
                                           $query2 = sprintf("delete from Block where
BloID=%s",$line1[0]);
                                           //print "\t\t\t<p class=\"3\">$query2</p>\n";
                                           $result2 = mysql_query($query2);
                                   }
                                   $query1 = sprintf("delete from Device where
DevID=%s",$line[0]);
                                   //print "\t\t<p class=\"2\">$query1</p>\n";
                                   $result1 = mysql_query($query1);
                           }
                           //smazani modelu
                           $query = sprintf("delete from Model where ModID=%s",$_GET['ID']);
                           //print "\t<p class=\"1\">$query</p>\n";
                           $result = mysql_query($query);

                   }
                   if(($_GET['Oper']==2)&&($rights==100)){
                           //echo "Povoleno mazani";
                           $delete = 1;
                   }

           $result = mysql_query("SELECT * FROM Model ORDER BY CreateDate, CreateTime");
           if (!$result) {
                   echo mysql_error();
                   exit;
           }
           // Zobrazení výsledku v HTML
                   print "<p><table widh=80%>\n";
                   print "\t<tr>\n";
                   print"\t\t<th>ID</th>\n";
                   print"\t\t<th>Model Name</th>\n";
                   print"\t\t<th>Created by</th>\n";
                   print"\t\t<th>Created</th>\n";
                   print"\t\t<th>Updated</th>\n";
```

```
                            print"\t\t<th>Valid until</th>\n";
                            if($delete) print"\t\t<th>Del</th>\n";
                            print "\t</tr>\n";
                            $i = 0;
                            while ($line = mysql_fetch_array($result, MYSQL_NUM)) {
                                //platnost modelu
                                $queryPlat = sprintf("SELECT ValidDate, ValidTime FROM Model
WHERE ModID=%d",$line[0]);
                                $resultPlat = mysql_query($queryPlat);
                                if (!$resultPlat) {
                                     echo mysql_error();
                                     exit;
                                }
                                else{
                                      if($linePlat = mysql_fetch_array($resultPlat, MYSQL_NUM))
{
                                                $DBdate="$linePlat[0] $linePlat[1]";
                                                $data_time_stamp = strtotime($DBdate);
                                                $comp_time_stamp =
mktime(date('H'),date('i'),date('s'),date('n'),date('d'),date('Y'));
                                                $PCdate = date("d.n.Y H:i:s",$comp_time_stamp);
                                                if($comp_time_stamp>$data_time_stamp){
                                                   $platnost=0;
                                                }
                                                else{
                                                     $platnost=1;
                                                }
                                        }
                                }

                                if($i % 2) $tag = "td class=1"; else $tag = "td class=2";
                                print "\t<tr>\n";
                                print "\t\t<$tag>$line[0]</$tag>\n";
                                print "\t\t<$tag>";
                                if(!$delete) print" <a
href=\"devices.php?Mod=$line[0]&LUID=$luid\">";
                                print "$line[8]</a></$tag>\n";
                                print "\t\t<$tag>$line[9]</$tag>\n";
                                print "\t\t<$tag>$line[1] $line[4]</$tag>\n";
                                print "\t\t<$tag>$line[2] $line[5]</$tag>\n";
                                print "\t\t<$tag>$line[3] $line[6]</$tag>\n";
                                if($delete && !$platnost)
                                   print "\t\t<$tag align=\"center\"><a
href=\"models.php?Oper=1&ID=$line[0]&LUID=$luid\"><img border=\"0\" src=\"images/delete.jpg\"
width=\"16\" height=\"16\"></a></$tag>\n";
                                else
                                    print "\t\t<$tag><Small>ONLINE</Small></$tag>\n";
                                print "\t</tr>\n";
                                $i++;
                            }
                            print "</table>\n";

                            // Odpojení z MySQL databáze
                            mysql_close($conn);
            ?>
</p>

</body>

</html>
```


Modely - Microsoft Internet Explorer
Soubor Úpravy Zobrazit Oblíbené Nástroje Nápověda
Zpět Hledat Oblíbené Historie
Adresa http://eprw2hp124/diplomka/models.php?Oper=DbLUID=1&V=19334
Přejít Odkazy
ČVUT Praha, FEL - Diplomová práce - Přístup ke sběrnici Profibus PA přes internet - Dušan Hlaváč
Modely
Navigace
Menu Nahoru Reload
Právě jsou v databázi dostupné následující modely:
ID Model Name Created by Created Updated Valid until
1 Testovací hlavacd 2002-12-13 21:15:09 2002-12-13 23:52:11 2013-12-20 23:52:11
2 Testovací hlavacd 2002-12-13 23:52:11 2002-12-13 23:52:11 2013-12-20 23:52:11
1289 Model vodarny DPC2_Server v1.0 2002-12-20 14:26:25 2002-12-20 14:26:25 2002-12-20 14:26:25
1290 Model vodarny DPC2_Server v1.0 2002-12-20 16:38:08 2002-12-20 16:38:08 2002-12-20 16:38:08
1292 Model vodarny DPC2_Server v1.0 2002-12-20 17:45:22 2002-12-20 17:45:22 2002-12-20 17:45:22
1293 Model vodarny DPC2_Server v1.0 2002-12-20 17:57:24 2002-12-20 17:57:24 2002-12-20 17:57:24
Místní síť intranet