České vysoké učení technické v Praze
Fakulta elektrotechická
Katedra řídicí techniky

Diplomová práce

Profinet

Kamil Vyštejn

kamil.vystejn@volny.cz

Praha 2003

České vysoké učení technické v Praze - Fakulta elektrotechnická

Katedra řídicí techniky **Školní rok:** 2000/2001

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Kamil Vyštejn

Obor: Technická kybernetika

Název tématu: ProfiNet

Zásady pro vypracování:

1. Seznamte se s objektovým přístupem k automatizaci podle specifikace ProfiNet.
2. Realizujte zařízení komunikující na ProfiNetu.
3. Navrhněte a realizujte automatizační systém s využitím profesionálního zařízení a zařízení podle bodu 2.

Seznam odborné literatury: Dodá vedoucí práce.

Vedoucí diplomové práce: Ing. Pavel Burget

Datum zadání diplomové práce: listopad 2000 (změna zadání: 11.09.2002)

Termín odevzdání diplomové práce: leden 2003

doc. Ing. Jiří Bayer, CSc.
vedoucí katedry

L.S.

prof. Ing. Vladimír Kučera, DrSc.
děkan v zast.

V Praze dne 16. 09. 2002

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu § 60 zákona č.121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne ……. …….
podpis

Poděkování

Chtěl bych touto cestou poděkovat Ing. Pavlu Burgetovi za vedení této práce, rodičům za podporu a Mozartovi za svěží a lehkou hudbu.

Abstrakt

Průmyslová síť Profinet je průmyslový standard vyvinutý sdružením PNO (PROFIBUS Nutzeorganisation) v roce 1999 jako následník sběrnice Profibus. Myšlenka Profinetu se opírá o stávající protokoly Ethernet, TCP/IP a RPC v rovině přenosové vrstvy, o DCOM v rovině aplikačního přístupu. Motivací vzniku této diplomové práce je vytvoření experimentálního modelu – průmyslové linky (2 dopravníky, robot, kontroler, operátorské stanoviště), na kterém bude možno prezentovat základní funkční vlastnosti (přenos hodnot, konfigurace). Základním prostředkem pro ověření funkčnosti přípravku je konfigurační nástroj – firma Siemens dodává program iMap, který ovšem nepracuje korektně s profinet klienty třetích stran. Proto bylo přikročeno k naprogramování vlastního konfiguračního nástroje, který je pojmenován KiMap a podporuje pouze konfigurační funkce.

Abstract

Profinet is an industry standard developed by PNO (PROFIBUS Nutzeorganisation) in 1999. Main principles of Profinet are based on open industry standards of Ethernet, TCP/IP, RPC at network layer, on DCOM at application layer. Goal of this diploma thesis is to make an experimental model of simple industry chain (2 movers, robot, controller, monitoring) on which we will be able to demonstrate basic functional behaviour (data transfer, configuration). As Siemens's configuration tool iMap is not able to setup third parties products there was developed another one which was named KiMap and supports basic configuration operations only.

Obsah

1. Úvod ... 1
2. Profinet ... 2
   2.1. Úvod ... 2
   2.2. RT model ... 3
      2.2.1. Koncept návrhu ... 3
      2.2.2. Runtime model ... 4
      2.2.3. Navigace mezi objekty PDev, LDev a RTAuto ... 5
      2.2.4. Komunikační model ... 6
      2.2.5. Tvorba přenosového paketu ... 6
   2.3. COM/DCOM ... 6
      2.3.1. Rozhraní IUnknown ... 8
      2.3.2. Rozhraní IDispatch ... 8
      2.3.3. Marshalling ... 9
   2.4. Diagnostika zařízení ... 10
      2.4.1. Společná diagnostika ... 11
      2.4.2. Detailní diagnostika ... 11
   2.5. Seznam rozhraní nad objekty PDev, LDev, RTAuto a ACCO ... 12
      2.5.1. CBAPhysicalDevice ... 12
      2.5.2. CBALogicalDevice ... 12
      2.5.3. CBAAcco ... 13
      2.5.4. CBARTAuto ... 13
   2.6. Operační stavy LDev ... 13
      2.6.1. Synchronní diagnostika ... 14
      2.6.2. Asynchronní diagnostika ... 14
   2.7. ACCO ... 14
      2.7.1. Optimalizace přenášených dat ... 16
      2.7.2. Věrohodnost přenášených dat ... 16
      2.7.3. Vyhodnocení QC ... 17
      2.7.4. Vytvoření a zánik datového spojení ... 17
      2.7.5. Ošetření chyb komunikace ... 18
      2.7.6. Podporované datové typy ... 19
   2.8. Profinet ES model ... 19
      2.8.1. iMap ... 20
   2.9. IDL ... 21
      2.9.1. Popis DCOM rozhraní ... 21
      2.9.2. Rozhraní definující properties ... 21
         2.9.2.1. Popis input property ... 22
         2.9.2.2. Popis output property ... 22
      2.9.3. Popis CBA komponenty v IDL ... 23
   2.10. Podpora IDL ve VisualC++ ... 24
   2.11. Úvod do XML ... 26
      2.11.1. Tag, element, atribut ... 26
      2.11.2. DTD šablony ... 27
      2.11.3. Popis CBA komponenty v XML ... 28

2.12. RS – profinet Runtime Source ... 32
2.12.1. Adresářová struktura RS ... 33
2.12.2. Významné soubory RS ... 33
2.12.3. Konfigurace Windows ... 34
2.12.3.1. Profinet server ... 35
2.12.3.2. Profinet klient ... 35
2.12.4. Profinet–server aplikace v prostředí Win32 ... 36
2.12.5. Návrh profinet–server aplikace ... 38
2.12.5.1. Definice LDev, RTAuto a properties – soubor sysobj.c ... 38
2.12.5.2. Hlavní program – main.c ... 40
2.12.6. Trasovací systém ... 42
2.12.6.1. Nastavení projektu ve VisualC++ ... 43
3. Popis experimentálního profinet modelu ... 45
3.1. Dopravník1 ... 48
3.1.1. Properties ... 48
3.1.2. Testovací módy ... 48
3.1.3. Popis programu ... 49
3.2. Dopravník2 ... 50
3.2.1. Properties ... 50
3.2.2. Testovací módy ... 50
3.2.3. Popis programu ... 51
3.3. Robot ... 52
3.3.1. Properties ... 52
3.3.2. Testovací módy ... 52
3.3.3. Popis programu ... 53
3.4. PLC ... 54
3.4.1. Properties ... 54
3.4.2. Hardwarová konfigurace ... 55
3.4.3. Mechanismus přenosu properties ... 55
3.4.4. Seznam interních proměnných ... 57
3.4.5. Popis programu ... 58
3.4.6. OB1: hlavní scan cyklus ... 59
3.4.7. OB35: periodický scan cyklus 100ms ... 60
3.4.8. FC50: teplotní ochrana Dopravníku1 ... 60
3.4.9. FC51: regulační algoritmus Dopravníku1 ... 61
3.4.10. FC52, FC53: teplotní ochrana, regulační algoritmus Dopravníku2 ... 61
3.4.11. FC54: regulační algoritmus pohybu robota ... 61
3.4.12. Export XML souboru ... 62
3.4.13. Download kódu v iMapu ... 62
3.5. Monitor ... 63
3.5.1. Properties ... 63
3.5.2. Popis programu ... 63
4. KiMap ... 65
4.1. Implementace EXPAT XML Parser knihovny ... 65
4.2. Struktura attribut_t ... 66
4.2.1. Popis položek a metod ... 66
4.3. Struktura xml_node_t ... 67

4.3.1. Popis položek a metod ... 67
4.4. Funkce build_xml_tree ... 68
4.5. Procedura find_xml_nodes ... 70
4.6. Variabilní datový typ VARIANT ... 71
4.7. Definice portu – struktura port_t ... 72
4.7.1. Popis položek a metod ... 72
4.7.2. „CBA XML“ konstruktor ... 73
4.7.3. „KIMAP XML“ konstruktor ... 73
4.8. Definice CBA komponenty – struktura cba_device_t ... 73
4.8.1. Seznam položek a metod ... 73
4.8.2. „CBA XML“ konstruktor ... 74
4.8.3. „KIMAP XML“ konstruktor ... 75
4.8.4. Metoda initCBA() ... 75
4.8.5. Metoda clearCBA() ... 76
4.9. Definice CBA spojení – struktura connection_t ... 76
4.9.1. Seznam položek a metod ... 77
4.10. KIMAP XML – ukládání KiMap projektu ... 77
4.11. Konfigurace a rušení datových spojení ... 78
4.11.1. ICBAAccoMgt::AddConnections ... 78
4.11.2. ICBAAccoMgt::RemoveConnections ... 79
4.11.3. ICBAAccoMgt::ClearConnections ... 79
5. Obsah CD ... 83
5.1. VisualC++ projekty ... 84
5.1.1. Workspace libraries.dsw ... 84
5.1.2. Workspace expat.dsw ... 84
5.1.3. Workspace diplomka.dsw ... 84
5.1.4. Workspace kimap.dsw ... 85
5.1.5. Workspace pst.dsw ... 85
5.1.6. Workspace CBAtest.dsw ... 85
5.2. Simatic Manager projekty ... 85
5.2.1. PLC ... 85
5.2.2. PLCT ... 85
6. Závěr ... 86
7. Literatura ... 87
8. Přílohy ... i

Seznam obrázků

Obrázek 2–1 Profinet – koncept návrhu ... 3
Obrázek 2–2 Runtime model a jeho obraz v návrhové fázi vývoje ... 4
Obrázek 2–3 Navigace PDev → LDev → RTAuto ... 5
Obrázek 2–4 Komunikační model Profinetu ... 6
Obrázek 2–5 Struktura přenosového paketu ... 6
Obrázek 2–6 RPC komunikace mezi procesy na různých strojích ... 7
Obrázek 2–7 Dědění rozhraní IUnknown ... 8
Obrázek 2–8 Volání metod pomocí IDispatch::Invoke ... 9

Obrázek 2–9 Způsob volání metody klientem a její spuštění na serveru ... 10
Obrázek 2–10 Společná diagnostika ... 11
Obrázek 2–11 Seznam všech rozhraní nad objekty PDev, LDev, RTAuto, ACCO ... 12
Obrázek 2–12 Stavový diagram LDev zařízení ... 13
Obrázek 2–13 LifeState kódy ... 14
Obrázek 2–14 Struktura ACCO ... 15
Obrázek 2–15 Vyhodnocení QC ... 17
Obrázek 2–16 Mechanismus přenosu dat ... 18
Obrázek 2–17 Ošetření chyb komunikace ... 18
Obrázek 2–18 Převod ES → RT model ... 20
Obrázek 2–19 Generování kódu pro automarhaller ... 25
Obrázek 2–20 ESMaster sekce ... 28
Obrázek 2–21 WWW odkaz ... 29
Obrázek 2–22 ESDevice sekce ... 29
Obrázek 2–23 ESAuto sekce ... 30
Obrázek 2–24 Connection sekce ... 30
Obrázek 2–25 Definice počtu portů ... 31
Obrázek 2–26 Definice portu ... 31
Obrázek 2–27 Vrstvy implementované v RS ... 32
Obrázek 2–28 Adresářová struktura RS ... 33
Obrázek 2–29 Profinet–server a profinet–klient ... 35
Obrázek 2–30 Vznik PDev a default–LDev ... 37
Obrázek 2–31 Jméno LDev ... 39
Obrázek 2–32 Počet RTAuto pro jednotlivá LDev ... 39
Obrázek 2–33 Celkový počet RTAuto ... 39
Obrázek 2–34 Jména jednotlivých RTAuto ... 39
Obrázek 2–35 Deklarace CBA properties ... 39
Obrázek 2–36 Inicializace CBA properties ... 40
Obrázek 3–1 Fyzikální schéma modelu ... 45
Obrázek 3–2 Elektrické schéma modelu ... 47
Obrázek 3–3 Dopravník1 ... 48
Obrázek 3–4 Dopravník1 – timer callback funkce ... 49
Obrázek 3–5 Dopravník2 ... 50
Obrázek 3–6 Dopravník2 – timer callback funkce ... 51
Obrázek 3–7 Robot ... 52
Obrázek 3–8 Robot – timer callback funkce ... 53
Obrázek 3–9 PLC ... 54
Obrázek 3–10 PLC – hardwarová konfigurace ... 55
Obrázek 3–11 PLC – mechanismus přenosu properties ... 57
Obrázek 3–12 PLC – OB1: hlavní scan cyklus ... 59
Obrázek 3–13 PLC – OB35: periodický cyklus 100ms ... 60
Obrázek 3–14 FC54: stavový diagram pohybu robota ... 61
Obrázek 3–15 Monitor ... 63
Obrázek 4–1 Struktura xml_node_t ... 67
Obrázek 4–2 XML strom ... 68

Seznam tabulek

Tabulka 2–1 QoS – Perioda přenosu dat ... 16
Tabulka 2–2 QC kódy ... 16
Tabulka 2–3 Podporované typy datového spojení ... 19
Tabulka 2–4 Druhy operace download ... 30
Tabulka 3–1 Dopravník1 properties ... 48
Tabulka 3–2 Dopravník1 – seznam testovacích módů ... 48
Tabulka 3–3 Dopravník2 properties ... 50
Tabulka 3–4 Dopravník2 – seznam testovacích módů ... 50
Tabulka 3–5 Robot properties ... 52
Tabulka 3–6 Robot – seznam testovacích módů ... 53
Tabulka 3–7 PLC properties ... 54
Tabulka 3–8 FC55: kódování stavů robota ... 62
Tabulka 4–1 attribut_t: položky a metody ... 66
Tabulka 4–2 xml_node_t: položky a metody ... 68
Tabulka 4–3 port_t: položky a metody ... 72
Tabulka 4–4 cba_device_t: položky a metody ... 74
Tabulka 4–5 connection_t: položky a metody ... 77

Seznam příkladů

Příklad 2–1 Vytvoření instancí PDev, LDev, RTAuto ... 5
Příklad 2–2 IDL popis DCOM rozhraní ... 21
Příklad 2–3 IDL popis DCOM rozhraní s properties ... 22
Příklad 2–4 IDL popis CBA komponenty ... 24
Příklad 2–5 idl.h s deklarací IDL rozhraní ve formě třídy ... 25
Příklad 2–6 Struktura XML dokumentu ... 26
Příklad 2–7 XML atributy ... 27
Příklad 2–8 DTD v XML souboru ... 27
Příklad 2–9 DTD v externím souboru ... 27
Příklad 2–10Popis PDev ... 38
Příklad 2–11 Popis LDev ... 38
Příklad 2–12 Počet LDev zařízení ... 38
Příklad 2–13 Manipulace s input/output properties ... 40
Příklad 4–1 Rozklad XML souboru ... 65
Příklad 4–2 Spojový seznam 4 atributů ... 66
Příklad 4–3 Zpracování jednoduchého XML souboru ... 69
Příklad 4–4 Funkce build_xml_tree ... 69
Příklad 4–5 Funkce StartElementHandler ... 70
Příklad 4–6 Funkce EndElementHandler ... 70

Seznam příloh

Příloha A Ukázka obsahu třídy IDLLib_Tbl ... i

Příloha B iMap – Plant view ii
Příloha C iMap – Network view iii
Příloha D Monitor – vizuální výstup iv
Příloha E Monitor – seznam klávesových zkratek iv
Příloha F PLC – příklad datového bloku DB100 v
Příloha G PLC – příklad použití funkčního bloku FB88 v
Příloha H PLC – příklad funkce „update input properties“ vi
Příloha I PLC – příklad funkce „update output properties“ vi
Příloha J PLC – mapování interních proměnných vii
Příloha K KiMap – hlavní dialogové okno viii
Příloha L KiMap – parametry komponenty viii
Příloha M KiMap – parametry spojení viii
Příloha N KiMap – výpis KiMap DTD šablony ix

1. Úvod

Průmyslová síť Profinet byla vyvinuta sdružením PNO (PROFIBUS Nutzeorganisation) v roce 1999 jako přímý následník Profibusu, který v současné době nedostačuje nárokům průmyslových řídicích systémů. Při návrhu je každé zařízení (čidlo, PLC) nebo program (běžící algoritmus) chápáno jako jedna komponenta, z pohledu profinetu platí: komponenta = proces vykonávající samostatnou činnost. Odtud také pochází jiné označení pro profinet – **Component Based Automation (CBA)**.

Cílem této diplomové práce je ověřit a otestovat základní funkční projevy tohoto nového standardu. Jako demostrace byl naprogramován jednoduchý model řídicího řetězce simulující výrobní linku – 2 dopravníky, robot, PLC a monitor (vizualizační nástroj).

Dopravníky, robot a monitor jsou aplikace simulující činnost skutečných zařízení – jsou naprogramovány jako Windows aplikace a využívají profinet Runtime Source (RS). RS je softwarový balík v jazyce C, který umožňuje vytvářet profinet komponenty v prostředí PC nebo přistupovat k jiným komponentám přes síťové rozhraní. Profinet RS je k dispozici pouze členům sdružení Profibus Int. PLC je typu CPU315–2DP rozšířené o kartu CP343–1PN, která poskytuje profinet rozhraní.

Z pohledu terminologie neexistuje pro některé termíny dostatečně výstižný český ekvivalent, proto je často využíváno původních anglických. Z důvodu jednoznačnosti a přehlednosti práce jsou anglické termíny využívány i v pasažích, kde to není nezbytně nutné. Jedná se zejména o tyto: device (technologické zařízení, jednotka), connection (datové spojení výstup→vstup), property (vstup nebo výstup), tag, element.

Protože profinet využívá několik softwarových standardů, se kterými nemusí být čtenář dostatečně seznámen, je každému standardu věnována jedna kapitola popisující jeho základní principy. Předpokládají se základní pojmy teorie průmyslových sítí, programovacího jazyka C, práce v prostředí VisualC++, znalost činnosti PLC řady Simatic–300 a návrhového prostředku Simatic Manager.

Kapitola 2 seznamuje čtenáře se specifikací profinetu, použitými softwarovými prostředky (DCOM, IDL, XML) a zároveň poskytuje podrobný návod jak naprogramovat profinet–server aplikaci . Kapitola 3 popisuje experimentální model výrobní linky a na konkrétním příkladu demonstruje způsob implementace profinet komponenty na PLC typu CPU315–2DP. Kapitola 4 popisuje aplikaci KiMap, která supluje nefunkční vlastnosti návrhového nástroje iMap.

Primárním zdrojem informací pro tuto práci je [1] a [2], kam je také často odkazováno.

2. Profinet

2.1. Úvod

Síť Profinet je snahou o vytvoření široce akceptovatelného prostředku pro řízení na vyšší úrovni abstrakce, než je tomu u ostatních průmyslových sítí. Základní vlastnosti sítě Profinet jsou:

- striktní oddělení návrhu od jeho implementace
- horizontální rozšiřitelnost mezi hardwarovými prostředky
- vertikální rozšiřitelnost mezi různými stupni řízení
- otevřenost pro již etablované průmyslové sítě – ty lze zavést jako podsíť profinetu pomocí proxy–serverů
- přenosovou vrstvou je ethernet (každé zařízení zapojené do sítě profinet je definováno unikátní IP adresou)
- komunikace probíhá pomocí protokolu TCP/IP
- přístup k jednotlivým hardwarovým zařízením je realizován podle standardu DCOM

Hlavním rysem profinetu je větší důraz na návrh struktury a funkce řídicího řetězce, než na jeho hardwarovou implementaci. Komponenty (objekty) s určitou funkcí jsou vkládány do návrhového inženýrského systému a logicky spojovány tak, jak to odpovídá požadavkům řízené technologie. Počítá se s tím, že výrobce vybavil dané zařízení sadou povinných rozhraní, definovaných v profinet normě. Každý automatizační objekt obsahuje následující množinu rozhraní:

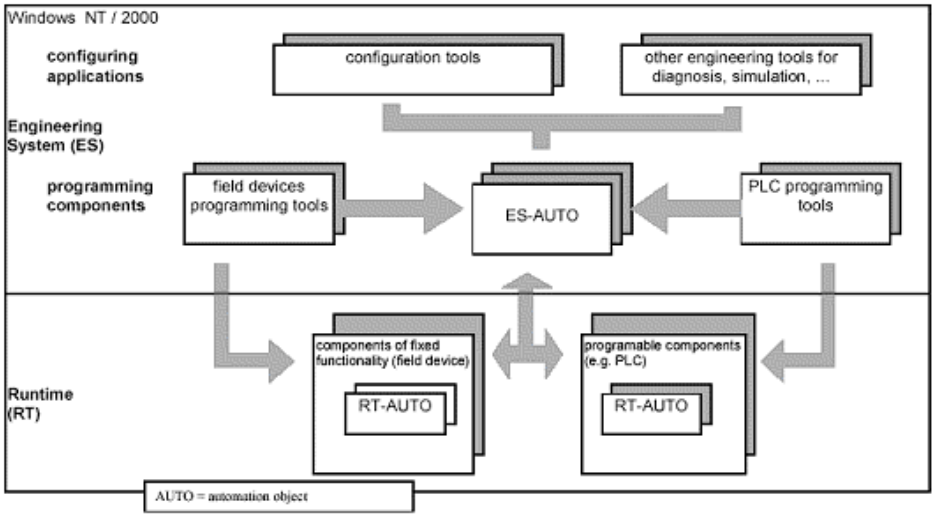
- povinná rozhraní podle profinet standardu
- volitelná rozhraní, která mohou ale nemusí být implementována; pokud jsou, musí splňovat definovanou specifikaci
- rozhraní využívající rozhraní ACCO (jakési „nadrozhraní“ pro funkce které nejsou známy v době návrhu zařízení, ale jejichž nutnost vyplyne během řešení inženýrského problému)

Pro interakci mezi komponentami řízeného procesu se využívá standardu DCOM firmy Microsoft. Ten definuje nad každou komponentou jedno nebo více rozhraní, pomocí nichž se přistupuje k funkcím a datům. DCOM definuje jak dané funkce vyvolávat, způsob jejich implementace uvnitř komponent zůstává skryt.

V každém regulační procesu se pracuje s naměřenými daty – profinet využívá několik prostředků pro jejich přenos – cyklické čtení, přenos hodnoty při definované změně, vkládání dodatečných informací (časové známky, měřítko důvěryhodnosti přenesené hodnoty – QualityCode). Profinet tedy zahrnuje podobné mechanismy jako ostatní průmyslové sítě.

2.2. RT model

2.2.1. Koncept návrhu



Obrázek 2.1 Profinet – koncept návrhu

Koncept profinetu od sebe odděluje oblast návrhu řídicího řetězce (Engineering System – ES) a jeho samotný běh na reálných zařízeních (Runtime – RT). Důvodem je zjednodušit proces návrhu tak, abychom se nemuseli zajímat o způsob implementace jednotlivých funkcí technologických zařízení – programátora obvykle zajímá, co dané zařízení umí a ne jak to dělá.

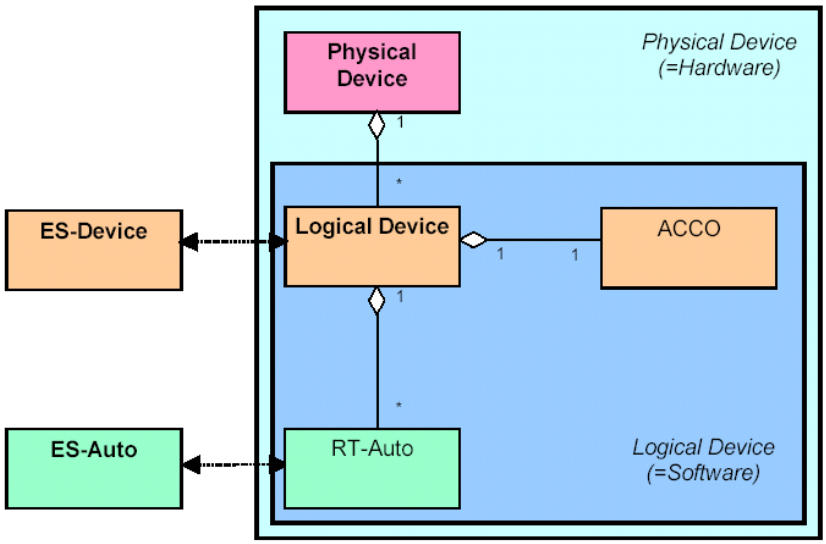
Objekt vykonávající určitou činnost se v terminologii profinetu označuje AUTO, přičemž RTAUTO představuje výkonný prvek navrhovaného systému (například PLC), ESAUTO je jeho obraz v návrhovém software (reprezentace programem, mapou vstupů a výstupů, …).

Každý objekt profinetu obsahuje podobně jako u ostatních průmyslových sítí data o své činnosti (teplota, tlak, …). Tato data se v terminologii profinetu označují **properties** a přístup k nim je realizován pomocí funkcí rozhraní. Pro uložení resp. čtení hodnoty property slouží funkce *put* resp. g*et*.

RTAuto se stává funkčním po provedení rekompilace kódu ESAuto v návrhovém software a provedení downloadu kódu.

2.2.2. Runtime model

Vztah mezi runtime objektem a jeho obrazy v návrhové fázi vývoje je následující:



Obrázek 2.2 Runtime model a jeho obraz v návrhové fázi vývoje

Physical Device (zkráceně **PDev**) představuje jedno konkrétní fyzické zařízení. Je identifikováno IP adresou a zajišťuje přístup k síti. Přímý přístup k PDev je možný přes rozhraní *ICBAPhysicalDevice.*

Na jednom PDev je definováno jedno nebo více logických zařízení **LogicalDevice** (zkráceně **LDev**). LDev reprezentuje softwarové vybavení a jeho různé verze. LDev poskytuje základní funkce jako je identifikace či diagnostika. Přístup k LDev je možný přes rozhraní *ICBALogicalDevice.*

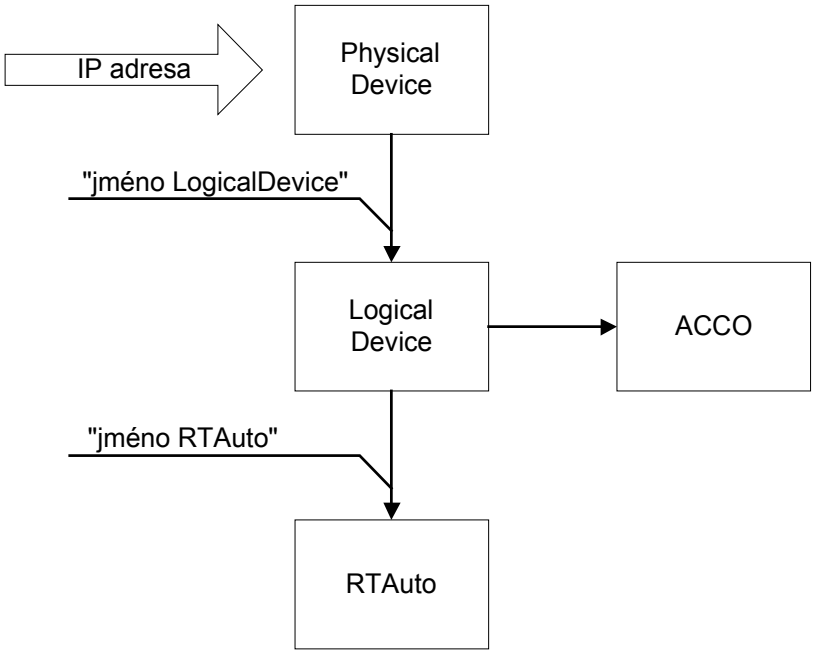
RTAuto představuje reálný hardwarový prostředek – např. PLC. Jedná se o zařízení, vykonávající samotnou činnost v regulačním řetězci, např. sensor, podavač atd.

ESDevice je model LDev v návrhovém software (reprezentován např. knihovnou v Simatic Manageru)

ESAuto je model RTAuto v návrhovém software (reprezentován např. STL programem v Simatic Manageru)

ACCO (Active Control Connection Object) vytváří datová a událostní spojení mezi RTAuto a provádí diagnostiku RTAuto.

2.2.3. Navigace mezi objekty PDev, LDev a RTAuto



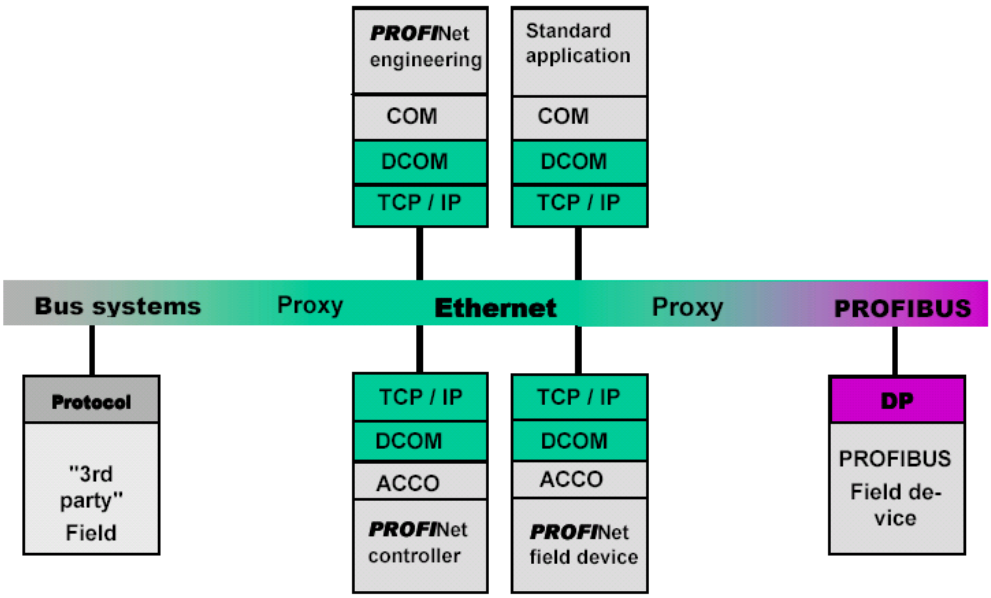
Obrázek 2.3 Navigace PDev → LDev → RTAuto

K objektu PDev se přistupuje pomocí IP adresy coby jedinečného identifikátoru. V rámci jednoho PDev se ke všem jeho LDev přistupuje pomocí jména, které jim bylo přiřazeno při vytvoření instance LDev v paměti. V rámci jednoho LDev se ke všem jeho RTAuto přistupuje obdobným způsobem. Příklad vytvoření instancí PDev, LDev a RTAuto v jazyce Visual Basic je uveden na následujícím příkladu.

```
Dim PDev As CBAPhysicalDevice
Dim LDev As CBALogicalDevice
Dim RTAuto As Object

' Create Object delivers the default interface (ICBAPhysicalDevice) of the physical device
'

Set PDev = CreateObject(„PhysicalDevice.PhysicalDevice“, „142.120.18.100“)

' Navigate to a specific logical device
Set LDev = PDev.LogicalDevice(„Firmware“)

' Navigate to a specific RT-Auto
Set RTAuto = LDev.RTAuto(„regulator 1“)
```

Příklad 2–1 Vytvoření instancí PDev, LDev, RTAuto

2.2.4. Komunikační model



Obrázek 2.4 Komunikační model profinetu

Nedá se říci, že profinet využívá pouze sítě Ethernet coby výhradní přenosové médium. Naopak – profinet je navržen tak, aby jej bylo možné aplikovat na co nejširší paletu průmyslových sběrnic. Při tomto přístupu je ale nutné najít určitý sjednocující prvek, který má sloužit jako jakási kotva určená pro adresaci jednotlivých komponent. Touto kotvou je IP adresa jako jednoznačný identifikátor daného zařízení. Přenosové médium tedy musí zaručit práci s IP adresou a TCP/IP protokolem jako nejnižší komunikační vrstvou. Ethernet obě tyto podmínky splňuje a proto je pro profinet sítí upřednostňovanou, nikoli povinnou. Pro přístup k cizím sítím je nutné zařadit mezi Ethernet a danou síť proxy router.

2.2.5. Tvorba přenosového paketu

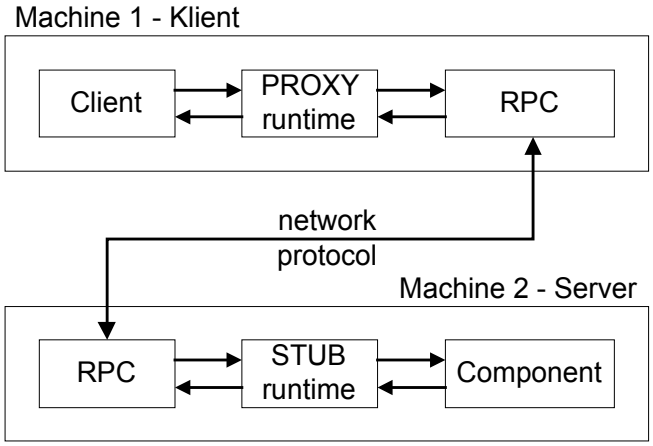
Do struktury přenosových paketů jsou vloženy následující informace:

a) ETHERNET hlavička
b) TCP/IP hlavička
c) RPC hlavička
d) DCOM hlavička (definice rozhraní a volané metody)
e) předávaná data

Obrázek 2.5 Struktura přenosového paketu

2.3. COM/DCOM

Protokol COM pochází z dílny firmy Microsoft. Umožňuje komunikaci procesů na jednom stroji; pro komunikaci mezi dvěma procesy na různých strojích je COM rozšířen o spolupráci s protokolem RPC (Remote Procedure Call). COM je založen na vztahu klient/server.



Obrázek 2.6 RPC komunikace mezi procesy na různých strojích

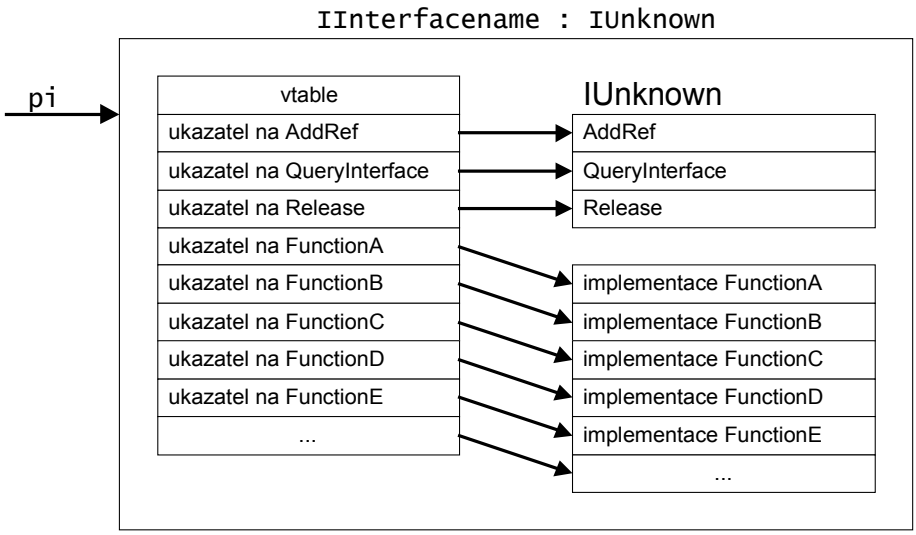
Data a metody, k jejichž výměně a volání má docházet, musí být na na obou počítačích zapouzdřeny ve formě komponenty, přičemž komponenta na straně klienta obsahuje metody typu PROXY, server obsahuje metody typu STUB.

- metoda PROXY – zajišťuje síťový přenos požadavku na volání určité funkce na straně serveru
- metoda STUB – při příchodu požadavku od klienta vykoná definovanou činnost a zajistí přenos výsledků zpět ke klientské PROXY funkci

Pokud chce klient vyvolat na straně serveru nějakou metodu X, musí nejdříve spustit lokálně metodu X_proxy, která zajistí přenos paketu na vzdálený stroj, kde jej přijme vrstva RPC. Ta po rozbalení paketu a identifikaci požadavku spustí požadovanou metodu X_stub, která vykoná svoji činnost a vrátí výsledky

Pro popis metod, které klient i server podporují, je zaveden pojem **rozhraní**. Rozhraní je soubor funkcí, které daná komponenta podporuje. Princip DCOMu záměrně vypouští property komponenty – DCOM pouze popisuje, jak je přenášet, jejich význam a obsahové určení jej „nezajímají“. DCOM rozhraní se popisují jazykem IDL viz kapitola 2.9.1. Rozhraní je implementováno jako objekt v paměti, nad kterým jsou definovány metody, které podporuje. **Každé rozhraní je identifikováno pomocí jedinečného identifikátoru IID (Interface ID).** Programově se ke službám rozhraní dostaneme pomocí ukazatele – pi (pointer to interface).

2.3.1. Rozhraní IUnknown



Obrázek 2.7 Dědění rozhraní IUnknown

IUnknown je mateřským rozhraním všech rozhraní, čili jakékoli rozhraní je od něj odvozeno a to buď přímo (*IInterface1*: *IUnknown*) nebo nepřímo (*IInterface2*: *IUnknown*, *IInterface3*: *IInterface2*).

Z důvodů zajištění konzistence komponenty v paměti (aby nedošlo k volání metody komponenty, která byla zrušena) je v *IUnknown* zaveden čítač odkazů – *reference_counter*. Princip funkce *reference_counteru* je následující:

1. při vytvoření instance rozhraní v paměti je *reference_counter* nastaven na 1
2. pokud nějaký proces má zájem o některou z metod rozhraní, zaregistruje se vyvoláním funkce ***IUnknown::AddRef()***, která inkrementuje hodnotu *reference_counteru*
3. pokud nějaký proces ztratí zájem o jím registrovanou metodu, vyvolá funkci ***IUnknown::Release()***, která dekrementuje hodnotu *reference_counteru*. Pokud je jeho hodnota nulová, provede se uvolnění instance rozhraní z paměti.

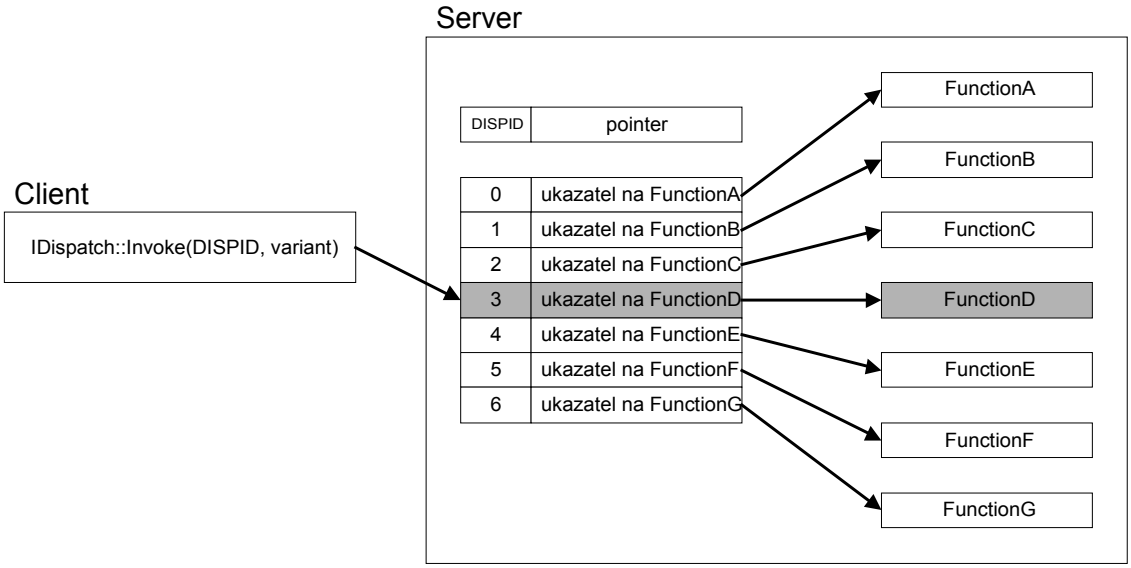
Instance rozhraní je v paměti reprezentována tabulkou ukazatelů (vtable – viz obrázek výše) na jednotlivé metody plus *reference_counter* plus informace o přístupových právech. Přístupová práva zde nebudou rozebírána, v profinetu nejsou využívána (nastavena na RPC_C_AUTH_LEVEL_NONE).

Třetí implementovanou metodou je ***IUnknown::QuerryInterface(IID)***. Tato metoda vrací ukazatel na rozhraní s daným IID. Používá se k „přepínání“ mezi různými rozhraními nad jednou komponentou.

2.3.2. Rozhraní IDispatch

Motivací k zavedení tohoto rozhraní je potřeba vyvolávat metody, které nejsou známy v době překladu komponenty. Rozhraní *IDispatch* je rozhraním jako každé jiné – je reprezentováno polem ukazatelů na metody a je zděděno od *IUnknown*. Oproti jiným rozhraním má *IDispatch* jednu zvláštní metodu – **Invoke**, která umí spustit jinou metodu. Tímto způsobem se metody vyvolávají nepřímo pomocí Invoke, jíž v argumentech uvedeme, kterou metodu chceme spustit a s jakými parametry. Aby byla spustitelná pomocí Invoke, musí se vyskytovat v poli metod rozhraní *IDispatch* a musí mít přiřazeno

identifikační číslo (dispatch identifier – DISPID). Mechanismus jejího volání přes Invoke ilustruje následující obrázek:



Obrázek 2.8 Volání metod pomocí IDispatch::Invoke

Při vyvolání metody se argumenty volané metody transportují skrze rozhraní *IDispatch* a poté se teprve dostávají k samotné metodě. Kód pro marshalling a unmarshalling (viz následující kapitola) se tedy týká pouze rozhraní *IDispatch*. Protože Invoke předem nezná volanou metodu ani typ jejích argumentů, musí se zavést univerzální datový typ – typ *variant.* Sám klient před tím, než zavolá metodu Invoke, zavede proměnnou *variant* a pomocí ní předá argumenty metodě. Variant definuje standardní tvar pro každý typ proměnné spolu s identifikátorem typu (short, integer atd.). Klient předá argument ve tvaru *variant* (což je obdoba marshallingu – tedy zabalení a rozbalení předávaných argumentů) spolu s DISPID metodě Invoke. Na straně serveru se variant rozbalí a použije ve volané metodě.

2.3.3. Marshalling

Protože klient i server mohou z principu DCOM běžet nejen na různých strojích, ale i platformách, je potřeba nějakým způsobem zajistit korektní předávání parametrů mezi metodami PROXY a STUB. Mechanismus předávání parametrů se označuje **marshalling**. Způsob jeho implementace je odvislý od programovacího jazyku a překladači. Existují dva druhy:

- standard marshalling – používán v C/C++
- universal marshalling (Type Library Marshalling) – používán ve Visual Basic, Javě

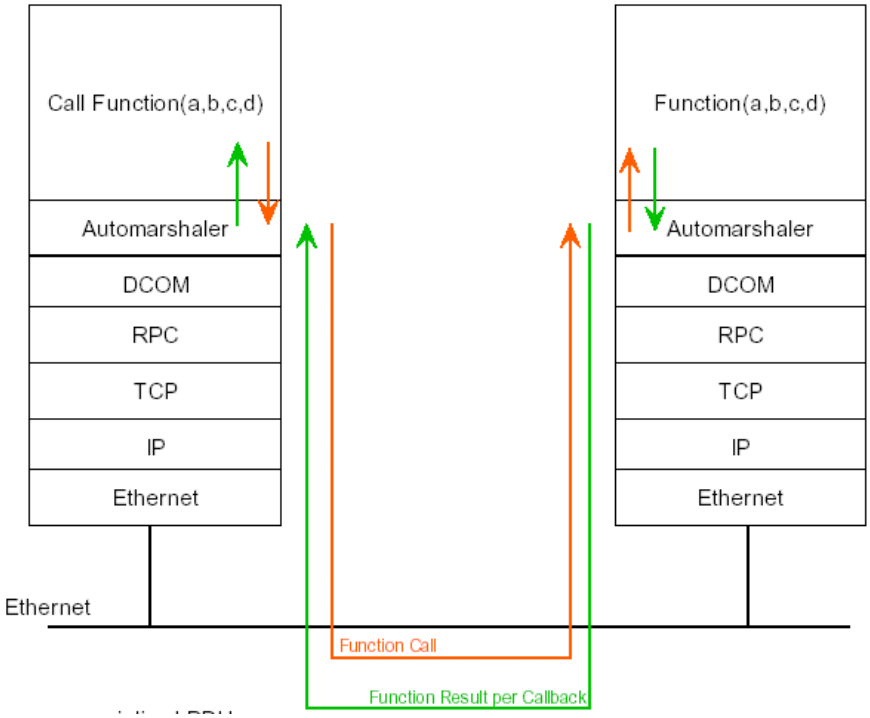
Rozdíl mezi oběma způsoby je pro naše účely nepodstatný – **v profinetu se používá universal marshallingu**. Ten využívá standardu RPC. Vrstvu zajišťující předávání parametrů budeme nadále označovat jako **automarshaller**.

Úlohou **automarshalleru** je tedy zapouzdřit parametry volající metody rozhraní do paketu, který je předán RPC vrstvě. Metoda, kterou požadujeme vyvolat na vzdáleném stroji, je uložena v DCOM hlavičce paketu. Při ukládání parametrů do paketu je třeba vzít v potaz, že předávány mohou být kromě standardních typů (char, int, long, …) také komplexní struktury, včetně ukazatelů. Ukazatele jsou

pointry na paměťové buňky na počítači, který danou metodu volá. V případě lokálního volání je situace jednoduchá, v případě vzdáleného volání funkcí je třeba rovněž přenést celou datovou strukturu, na kterou ukazatel ukazuje.

Automarshaller je implemetován na obou stranách – u klienta i u serveru. Zajišťuje následující služby:

- zapouzdření parametrů – konverze lokálních dat do paketu (marshalling)
- rozbalení parametrů – konverze paketu do lokálních dat (unmarshalling)
- tvorba seznamu použitých metod a parametrů pro debug účely



Obrázek 2.9 Způsob volání metody klientem a její spuštění na serveru

Automarshaller musí „zakrýt“ i takové rozdíly, jako je různá implementace typů na rozdílných platformách a překladačích (např. int je pod MS Visual Studio chápán jako 32–bitové číslo, pod SunOS jako 64–bitové), jejich různá reprezentace (little–endian resp. big–endian), způsob ukončování řetězců, atd.

2.4. Diagnostika zařízení

Způsoby vyvolání diagnostické informace jsou:

- synchronně cyklickým dotazováním (polling)
- asynchroně pouze při změně stavu (event diagnosis)

Způsoby hodnocení stavu zařízení:

- společná diagnostika (common diagnosis)
- detailní diagnostika (detailed diagnosis)

2.4.1. Společná diagnostika

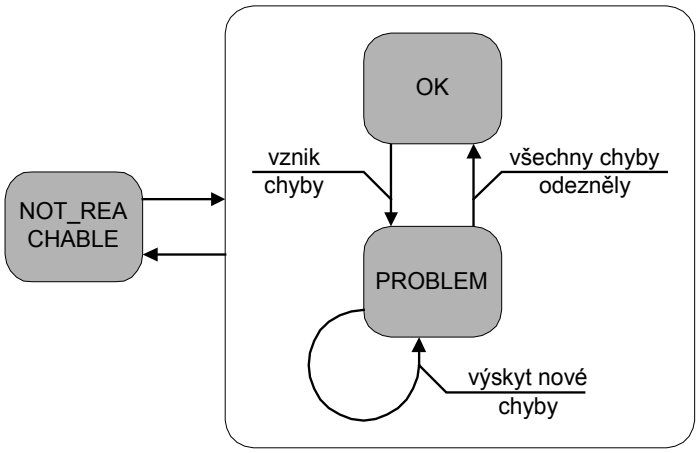
Je určena pro nejjednodušší diagnostiku zařízení – rozeznává pouze tři stavy:

- NOT_REACHABLE – zařízení není dostupné (chyba komunikace)
- OK – zařízení pracuje správně
- PROBLEM – vnitřní chyba zařízení

Při synchronním způsobu vyvolání detailní diagnostiky rozhraním *ICBAGroupError* je volajícímu procesu předán současný stav zařízení; v případě chyby PROBLEM je navíc předána hodnota *cookie*. Pokud je při dalším voláním hodnota *cookie* stejná, nedošlo k výskytu jiné chyby, jinak ano. Tento způsob de facto simuluje asynchronní vyhodnocení stavové informace.

Postup, jakým diagnostikované zařízení určuje hodnotu *cookie* není definován a platí: **Z hodnoty *cookie* nelze vyvodit žádné závěry o povaze chyby.** K tomuto účelu je třeba využít detailní diagnostiku.

Při asynchronním voláním diagnostiky rozhraním *ICBAGroupErrorEvent* jsou indikovány změny stavu OK → PROBLEM, PROBLEM → PROBLEM (výskyt nové chyby) a PROBLEM → OK:



Obrázek 2.10 Společná diagnostika

Chyba typu NOT_REACHABLE nemůže být ze své povahy (ztráta komunikace) vyhodnocena.

2.4.2. Detailní diagnostika

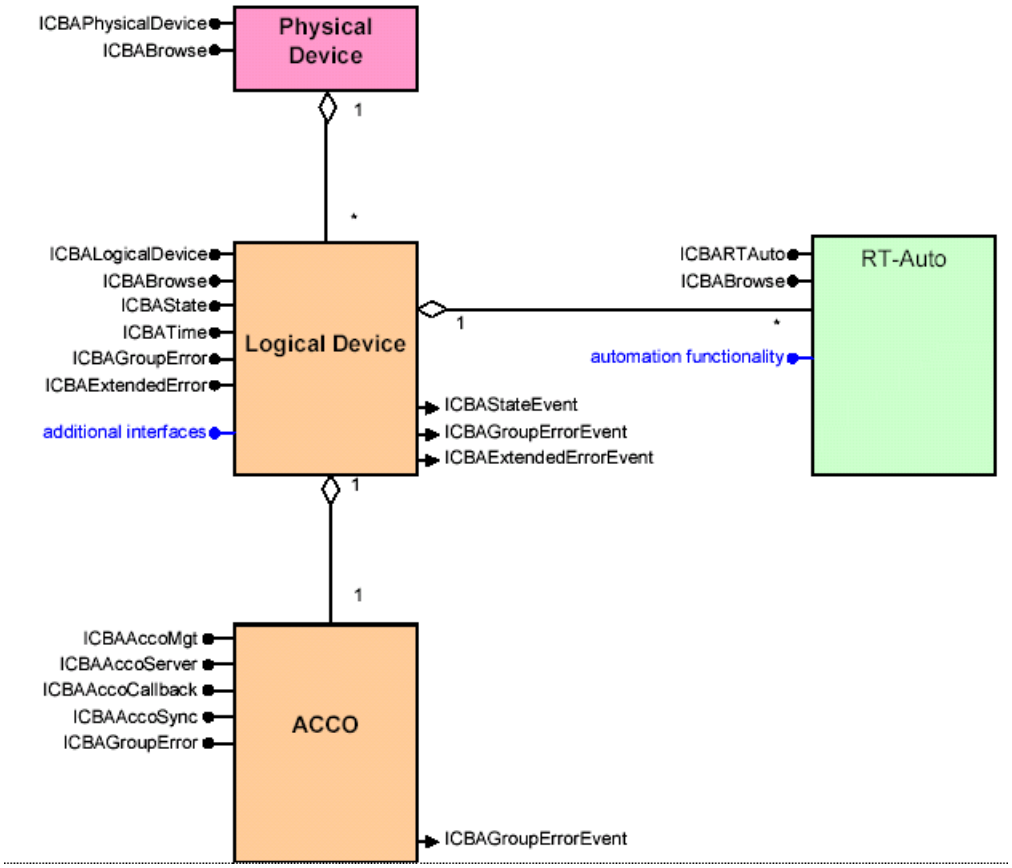
Chyby, které mohou nastat mohou být hardwarové – chyba paměti, zkratování vstupů, přetížení výstupů, softwarové – dělení nulou, volání neplatné procedury apod. Jejich reprezentace se může navíc zařízení od zařízení lišit (cyklický buffer, zásobník, …). Výčet všech možných chybových stavů na všech typech zařízení by čítal stovky, možná tisíce položek, proto je způsob vyhodnocení detailní diagnostiky záležitostí inženýrského modelu a není v profinet specifikaci podrobněji rozebrán.

Přesto jsou zavedeny alespoň dva druhy zpráv přenášejících diagnostickou informaci:

- **Brief information** – popis chyby ve tvaru prostého textu (například pro zobrazení na displeji) např. „`CP 312 – Memory fault error`“
- **Long Information –** přenos úplné chybové informace, kterou musí řídicí algoritmus dekódovat; strukturu této chybové informaci musí určit výrobce.

Synchronní a asynchronní přenos detailní diagnostické informace poskytuje rozhraní *ICBAExtendedError* a *ICBAExtendedErrorEvent*.

2.5. Seznam rozhraní nad objekty PDev, LDev, RTAuto a ACCO



Obrázek 2.11 Seznam všech rozhraní nad objekty PDev, LDev, RTAuto, ACCO

2.5.1. CBAPhysicalDevice

ICBAPhysicalDevice – identifikace hardwaru
ICBABrowse **–** vrátí seznam všech LDev

2.5.2. CBALogicalDevice

ICBALogicalDevice – identifikace softwaru a navigační kotva pro RTAuto a ACCO
ICBABrowse **–** vrátí seznam všech RTAuto
ICBAState – synchronní čtení a změna operačního stavu LDev
ICBAStateEvent – asynchronní vyvolání události při změně stavu ve všech rozhraních, která se předtím zaregistrovala metodou *ICBAState::AdviseState*
ICBATime – práce s časem (nepovinné rozhraní)
ICBAGroupError – diagnostika chyb
ICBAGroupErrorEvent **–** asynchronní vyvolání události ve všech rozhraních, která se předtím zaregistrovala metodou *ICBAGroupError::AdviseGroupError*

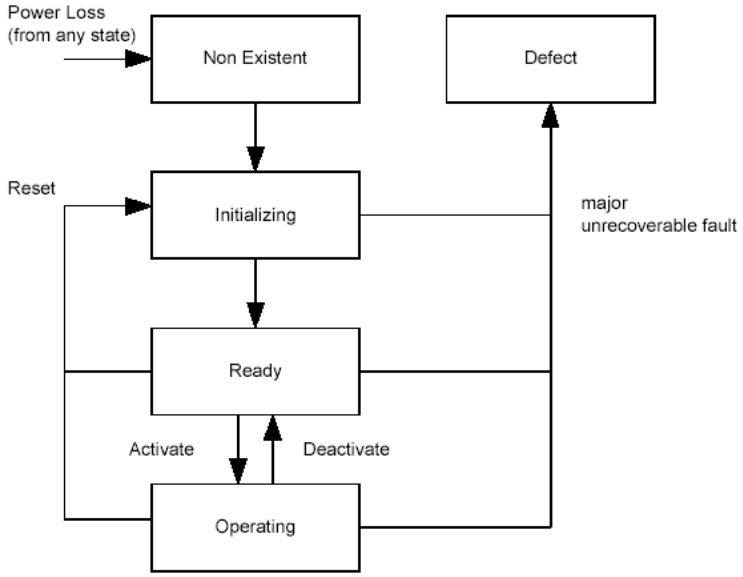
2.5.3. CBAAcco

ICBAAccoMgt – práce s konfigurační databází (tvorba a rušení ACCO spojení)
ICBAAccoCallback – asynchronní vyvolání události na straně consumera po příjmu dat
ICBAAccoServer – práce s konfigurační databází providera
ICBAAccoSync – synchronní čtení/zápis property
ICBAGroupError – diagnostika chyb
ICBAGroupErrorEvent **–** asynchronní vyvolání události ve všech rozhraních, která se předtím zaregistrovala metodou *ICBAGroupError::AdviseGroupError*

2.5.4. CBARTAuto

ICBARTAuto **–** identifikace
ICBABrowse **–** vrátí seznam všech properties

2.6. Operační stavy LDev



Obrázek 2.12 Stavový diagram LDev zařízení

Non Existent – pseudostav – zařízení není napájeno, proto nemůže komunikovat s okolím; uvádí se jen pro úplnost.
Defect – nastala chyba v PDev nebo LDev, pravděpodobně chyba v hardwaru PDev nebo softwaru LDev. Je nutný vnější zásah – jediná cesta jak zrestartovat LDev je vypnout/zapnout napájení. V závislosti na typu chyby může probíhat částečná komunikace s LDev.
Initializing – inicializace LDev po zapnutí resp. resetu
Ready – LDev je připraveno komunikovat. RTAuto je v klidovém stavu, výstupy jsou v definované hodnotě
Operating – provozní činnost LDev i RTAuto

Zařízení vykazuje činnost pouze ve stavu **Operating**. V ostatních stavech je v nečinnosti. Některá RTAuto poskytují informaci o svém stavu pomocí výstupní property LifeState která je typu unsigned char a její kódování je následující:

NonExistent	0x00
Initializing	0x01
Ready	0x02
Operating	0x03
Defect	0x04

Obrázek 2.13 LifeState kódování

Pro diagnostiku a monitorování stavu LDev jsou určena rozhraní *ICBAState* a *ICBAStateEvent*. *ICBAState* zajišťuje synchronní diagnostiku stavu, *ICBAStateEvent* asynchronní detekci změny stavu.

2.6.1. Synchronní diagnostika

ICBAState::get_State – vrátí aktuální stav LDev zařízení
ICBAState::Activate – přepne LDev do stavu Operating, pokud je to možné
ICBAState::Deactivate – přepne zařízení do stavu Ready, pokud je to možné
ICBAState::Reset – Reset LDev zařízení (přechod do stavu Initializing)

2.6.2. Asynchronní diagnostika

V dalším popisu se bude pro přehlednost jako *vzdálené* označovat zařízení na kterém dochází ke změně stavu, jako *lokální* zařízení na kterém chceme tuto změnu vyhodnotit.

Lokální obslužná rutina, která se má vykonat při změně stavu, se musí nejdříve u vzdáleného zařízení zaregistrovat pomocí funkce *ICBAState::AdviseState*. Jako parametr předá odkaz na sebe sama – vzdálené zařízení si tento odkaz uloží do vnitřního registru. Pokud u vzdáleného zařízení dojde ke změně stavu, vyvolá vzdálenou metodu *ICBAStateEvent::SendState*, která z registru načte odkaz na lokální obslužnou rutinu, vyvolá ji a předá jí hodnotu minulého a současného (nového) stavu.
Pro zrušení registrace slouží funkce *ICBAState::UnadviseState*.

2.7. ACCO

ACCO zajišťuje přenos dat a událostí mezi různými RTAuto. Umožňuje dynamicky vytvářet a rušit datová i událostní spojení za chodu RTAuto. Není tedy nutné provést rekompilaci kódu RTAuto po každé změně ve struktuře regulované soustavy. Toto je jedna z výrazných výhod profinetu proti ostatním průmyslovým sítím.

ACCO plní následující funkce:

a) vytváří a ruší kanály
b) přenáší data v daných časových intervalech (**QoS** – Quality of Service) s kvalitativní známkou (**QC** – Quality Code)
c) vyvolává událostí
d) implementuje obsluhu chybových stavů
e) optimalizuje přenosu dat

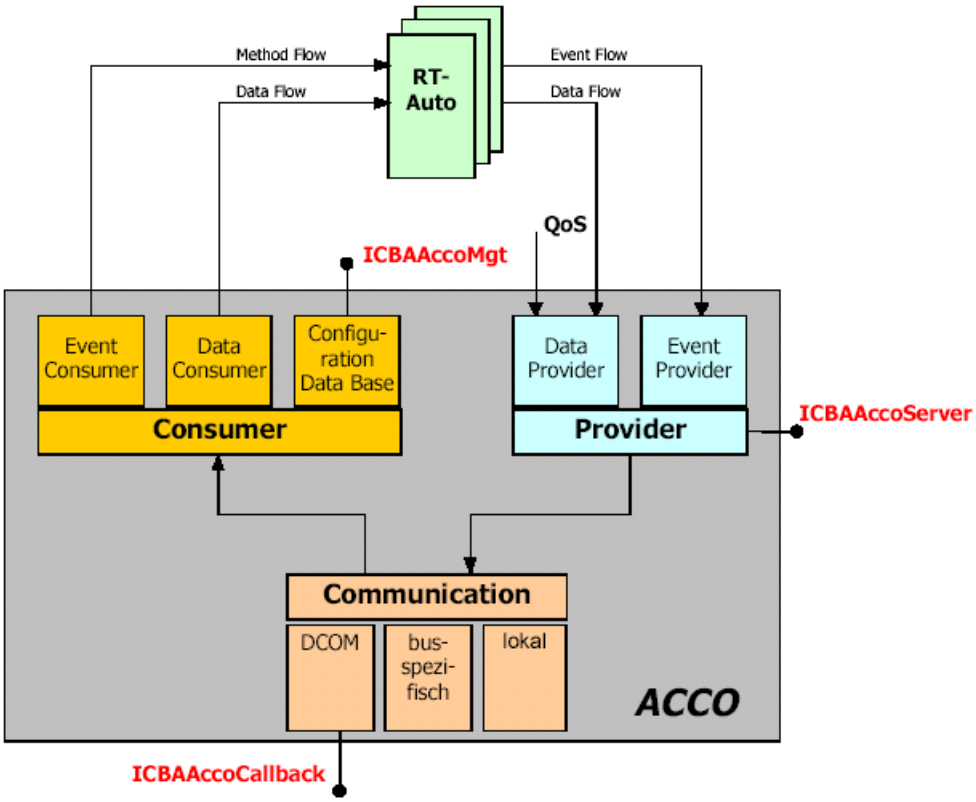
Pro navázání spojení jsou potřeba dva objekty RTAuto:

- producent dat – **provider**
- konzument dat – **consumer**

U provider/consumer spojení je iniciátorem přenosu dat provider! Jedná se tedy o opačný postup než u spojení typu klient/server.

Předmětem spojení musí být prvek stejného typu pro providera i consumera:

- pro **datové spojení** musí být data stejného typu. Pro providera musí mít data IDL atribut [propget]. Pro consumera musí mít data IDL atribut [propput].
- pro **událostní spojení** musí obě metody mít shodné parametry (tj. musí být stejného typu a stejné orientace [in] a [out])



Obrázek 2.14 Struktura ACCO

Pro registraci navázaných spojení má provider i consumer vlastní konfigurační databázi. K ní lze přistupovat pomocí rozhraní *ICBAAccoMgt* a tak vytvářet nebo rušit spojení. Pomocí rozhraní *ICBAAccoServer* lze měnit parametry již navázaného spojení. Rozhraní *ICBAAccoCallback* obsahuje jedinou funkci *OnDataChanged*, jíž provider vyvolá v consumerovi při přenosu hodnoty. Synchronní čtení/zápis properties definuje rozhraní *ICBAAccoSync*.

Pro každé spojení je v konfigurační databázi o partnerovi komunikace identifikační záznam, jména PDev a LDev je uložena ve tvaru prostého textu „`PDev!LDev`", v rámci LDev je identifikace jednotlivých RTAuto a jejich properties uložena rovněž ve tvaru prostého textu „`RTAuto.property`".

V aktuální verzi specifikace (V1.0) je popsán pouze postup pro vytváření datových spojení, událostní spojení není prozatím podporováno.

2.7.1. Optimalizace přenášených dat

ACCO zamezuje zbytečnému přetěžování sítě následujícími prostředky:

- **QoS** – Quality of Service – provider s danou periodou cyklicky testuje danou proměnnou a pokud je mimo stanovený rozsah (definovaný pomocí epsilon_value), zašle ji consumerovi

QoS	QoSType	QoSValue
cyklicky každých n milisekund	0	počet msec
cyklicky každých n sekund	1	počet sec
na vyžádání	10	–

Tabulka 2–1 QoS – Perioda přenosu dat

- **epsilon_value** – přenos nastane pouze pokud se proměnná změnila od minulého čtení o více než epsilon_value, tedy pokud:

$$abs(current_value - last_transfered_value) > epsilon_value$$

- v rámci daného spojení provider–consumer se zasílají všechna data se stejnou QoS najednou v jediném telegramu – zamezení fragmentace přenášených dat

2.7.2. Věrohodnost přenášených dat

Přenášená data se hodnotí pomocí 8–bitového čísla **QC (Quality Code)**. Podle jeho hodnoty lze považovat hodnotu za korektní či nikoliv. Celý 8–bitový rozsah je rozdělen do 5 skupin:

Quality	Status	Popis
Good (NC)	OK	hodnotu lze bez obav použít
Uncertain	last_usable_value	použití hodnoty z minulého čten last_usable_value
Uncertain	substitute_value	použití substitute_value
Uncertain	initial_value	byla načtena inicializační hodnota
Bad	Nepoužitelná	komponenta nepracuje správně

Tabulka 2–2 QC kódy

Podrobný výčet QC kódů viz. [1] kapitola 2.13.6.9.13.

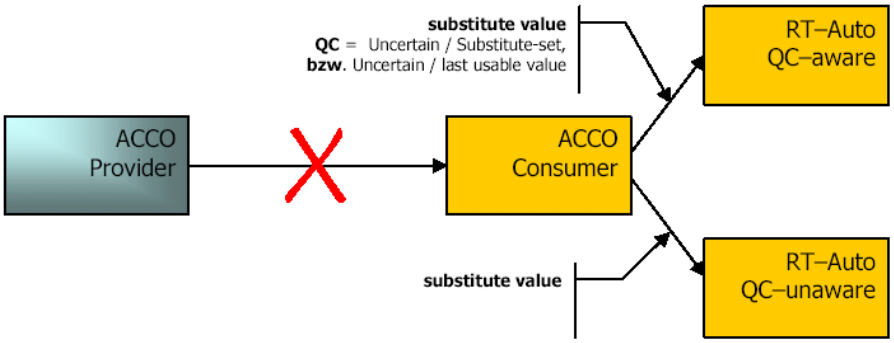
Pokud během komunikace nastane chyba, může se v závislosti na QC použít jedna z následujících dvou hodnot:

- **substitute_value** – předem definovaná „korektní“ hodnota (statisticky nejpravděpodobněji přenášená hodnota)
- **last_usable_value –** poslední korektně načtená hodnota

2.7.3. Vyhodnocení QC

RTAuto buď QC podporuje (QC–aware) nebo nikoli (QC–unaware). Pokud během komunikace dojde k chybě nebo zrušení spojení, pak:

a) QC–aware RTAuto vyhodnotí QC a podle jeho hodnoty dosadí buď *substitute_value* nebo *last_usable_value*
b) QC–unaware RTAuto dosadí vždy *substitute_value*.

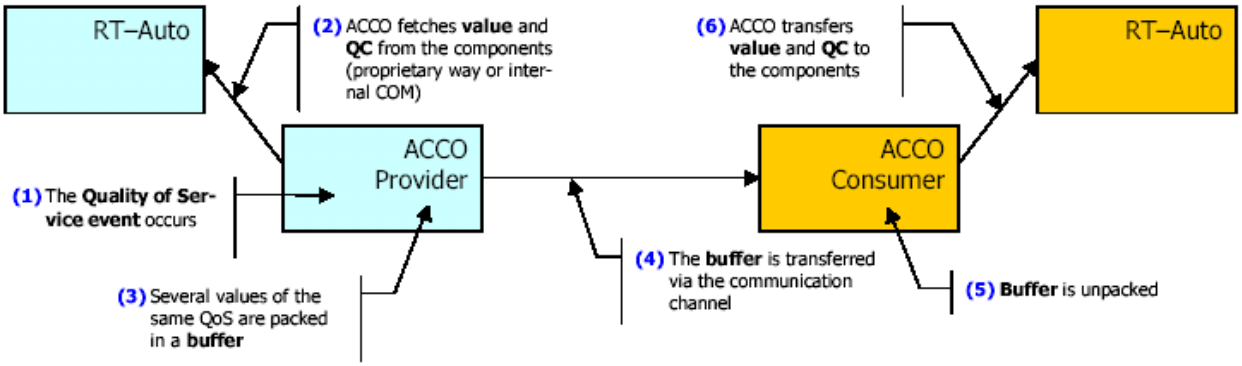


Obrázek 2.15 Vyhodnocení QC

2.7.4. Vytvoření a zánik datového spojení

Typický postup vytvoření datového spojení je následující:

1. řídicí proces zažádá consumera o vytvoření datového spojení vyvoláním metody *ICBAAccoMgt::AddConnections* a předá mu parametry (QoS, substitute_value, epsilon_value)
2. consumer zažádá providera o vytvoření spojení vyvoláním jeho metody *ICBAAccoServer::Connect* a předá mu tytéž parametry, které obdržel v bodě 1.
3. provider v daných časových intervalech (určených hodnotou QoSValue) testuje, zda se požadovaná proměnná nachází mimo interval definovaný pomocí epsilon_value. Pokud ano, přenese tuto hodnotu a QC vyvoláním metody *ICBAAccoCallback::OnDataChanged* na straně consumera.
4. řídicí proces zažádá consumera o zrušení spojení voláním metody *ICBAAcco::DeleteConnections*
5. consumer informuje providera o zrušení spojení vyvoláním metody *ICBAAccoServer::DisconnectMe*

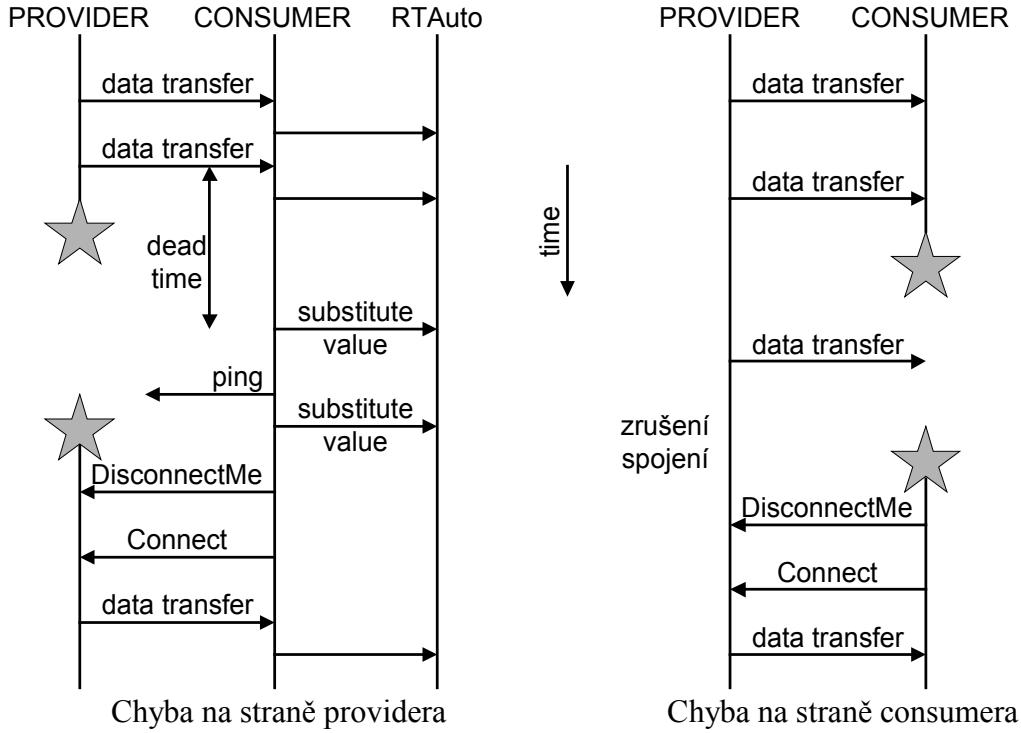


Obrázek 2.16 Mechanismus přenosu dat

2.7.5. Ošetření chyb komunikace

Pokud dojde při komunikaci k chybě na straně providera, pak consumer zruší všechny datová spojení s tímto providerem. Místo přenášených dat dosazuje *substitute_value* nebo *last_usable_value*, záleží na implementaci. Zároveň v pravidelných časových intervalech testuje zdali je provider přístupný. Jakmile obdrží pozitivní ping, pak přikáže providerovi zrušit všechna datových spojení (toto je ochrana proti situaci, kdy provider nedetekuje vlastní chyby). Poté jsou všechna datová spojení obnovena standardním postupem.

Pokud nastane chyba na straně consumera, pak provider „zapomene“ všechny datové spoje s tímto consumerem. Jakmile se consumer opět dostane k síti, pak sám automaticky požádá providera o zrušení všech spojení, protože během výpadku komunikace prováděl činnost popsanou v předešlém odstavci. Protože provider všechna spojení již zrušil, tak na tento požadavek (DisconnectMe) jednoduše nijak nereaguje.



Obrázek 2.17 Ošetření chyb komunikace

2.7.6. Podporované datové typy

V následující tabulce je uveden seznam datových typů, které jsou podporovány při přenosu datovým spojením. Protože datové typy se liší svým názvem v různých programových prostředcích, je uveden seznam datových typů pro aplikace, které byly využity v diplomové práci.

kód	Popis	OLE datatype	MIDL datatype	VisualC++ datatype	S7 datatype
2	2–byte signed int	VT_I2	short	short int	INT
3	4–byte signed int	VT_I4	long	long int	DINT
4	4–byte real	VT_R4	float	float	REAL
5	8–byte real	VT_R8	double	double	
6	currency	VT_CY	CY	CURRENCY	
7	date	VT_DATE	DATE	DATE	DT
8	Basic String	VT_BSTR	BSTR	_bstr_t	STRING
11	boolean	VT_BOOL	VARIANT_BOOL	bool	BOOL
16	1–byte signed int	VT_I1	char	char	CHAR
17	1–byte unsigned int	VT_UI1	unsigned char	unsigned char	BYTE
18	2–byte unsigned int	VT_UI2	unsigned short	unsigned short int	WORD
19	4–byte unsigned int	VT_UI4	unsigned long	unsigned long int	DWORD

Tabulka 2–3 Podporované typy datového spojení

VT_CY je přenášen jako 8–byte signed int; představuje cenovou jednotku bez specifikace měny vynásobenou 10000 a s rozlišením 15 celočíselných cifer, 4 desetinné cifry. Např. údaj 1325000 představuje cenu 132,50.

VT_DATE je přenášen jako 8–byte real a vyjadřuje čas v rozmezí 1.1.100 až 31.12.9999 s přesností na nanosekundy. Celočíselná hodnota reprezentuje počet dní od 30.12.1899 (dřívější data jsou reprezentována záporně), desetinná část reprezentuje část dne jako zlomek 24 – hodinového cyklu. Např. 5.50 = 4.1.1900 12:00.

VT_BOOL je přenášen jako signed short int, false = 0 (všechny bity „0“), true = –1 (všechny bity „1“)

Podrobně viz:

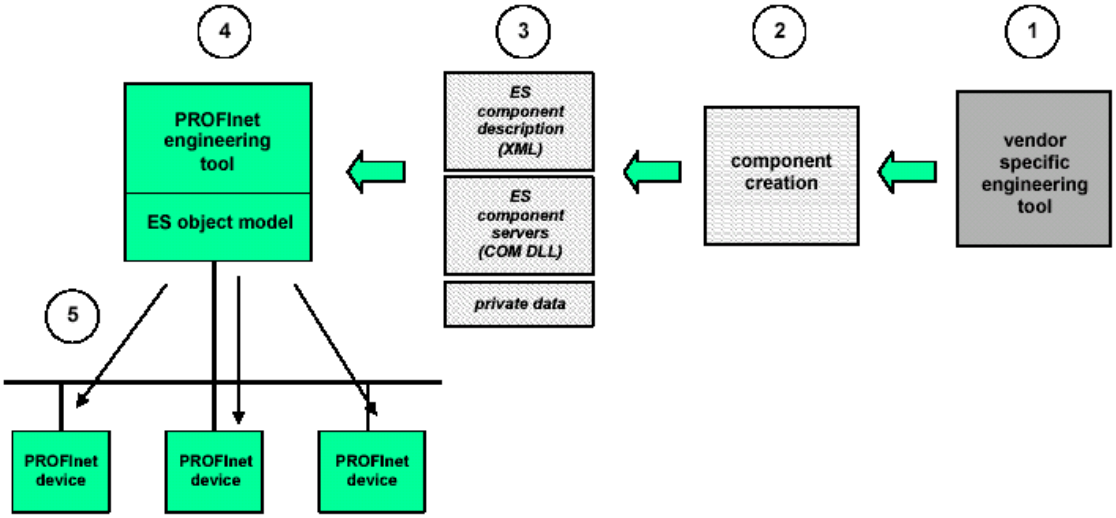
- [1] kapitoly 2.7, 2.14.6.4, 2.15.5.2
- [1] strana 3–31
- [5] téma „VARIANT and VARIANTARG“
- hlavičkový soubor wtypes.h

2.8. Profinet ES model

Postup při tvorbě regulačního řetězce by měl být následující:

1. profinet komponentu lze vytvořit v návrhovém ES software který podporuje export komponenty do XML souboru
2. naprogramovat a nakonfigurovat zařízení

3. vytvořit komponentu – převod kódu a definice vstupů/výstupů do tvaru, s kterým umí pracovat ES software tj. do XML. Kód komponenty nemusí být v XML – může se jednat např. o SimaticManager projekt. V tom případě musí být v XML specifikována cesta k projektu + odkaz na aplikaci, která umí download provést.
4. vložení komponenty do ES software – definice propojení s okolními komponentami
5. download ES popisu komponenty do reálného zařízení – z ESAuto se stává RTAuto



Obrázek 2.18 Převod ES → RT model

Uvedený postup je možné provést v programovacích prostředcích SimaticManager a iMap firmy Siemens:

Krok 1: naprogramujeme činnost zařízení v Simatic Manageru
Krok 2: vygenerujeme XML soubor
Krok 3: v iMapu načteme XML soubor a v grafické reprezentaci propojíme vstupy a výstupy komponenty s ostatními komponentami
Krok 4: provedeme download

Poznámka: Mezi ES a RT objektem je vždy relace 1:1. Tím je zaručeno, že regulační proces bude mít shodnou strukturu jako má návrh v ES softwaru.

2.8.1. iMap

iMap je základním konfiguračním nástrojem profinetu. Úkony které by měl vykonávat jsou:

1. jako vstup načítat XML popis komponent
2. nastavit IP adresu komponenty
3. umět pracovat s více LDev a RTAuto na jedné adrese (tj. relace PDev:LDev 1:N, relace LDev:RTAuto 1:M)
4. definovat spojení mezi komponentami v grafickém prostředí
5. u connections definovat parametry QoS, substitute_value, epsilon_value
6. provést download těchto connections
7. provést download kódu komponenty, pokud toto komponenta podporuje

8. zobrazovat online hodnoty connections (primitivní vizualizace)
9. provést porovnání online/offline kódu komponenty

Bohužel je nutné konstatovat, že iMap verze trial 1.1 dodaný katedře firmou Siemens ovládá pouze úkony 1. , 2. , 4. , 7. , u bodu 5. lze definovat pouze QoS.

Z pohledu tématu této diplomové práce se jako největší nedostatek jeví nemožnost provedení download connections, protože bez této vlastnosti nelze vůbec vytvořit regulační řetězec! Tato skutečnost byla důvodem pro naprogramování vlastního konfiguračního nástroje KiMap, který s použitím RS bez potíží ovládá načítání XML, definici datových spojení (včetně QoS, substitute_value, epsilon_value) a jejich download.

Podrobný popis aplikace KiMap je uveden v kapitole 4.

2.9. IDL

2.9.1. Popis DCOM rozhraní

IDL (Interface Definition Language) má po jazykové stránce velmi blízko k syntaxi C++:

```
[
 object,
 uuid(32bb8323-b41b-11cf-a6bb-0080c7b2d682),
 helpstring("Interface Example")
]
interface IExample : IUnknown
{
 HRESULT foo([in] int x, [in,out] int* y, [out] int* z);
 HRESULT goo([in] long size);
};
```

Příklad 2–2 IDL popis DCOM rozhraní

uuid – definuje iid – 128 bitový jedinečný identifikátor rozhraní. Pro jeho generování je v MS Visual Studio určena aplikace guidgen.exe. **Algoritmus výpočtu zaručuje jedinečnost v čase i prostoru** (vstupem algoritmu jsou čas, sériové číslo procesoru a náhodně vygenerované číslo).

helpstring – pomocný textový popis; nemá vliv na funkci rozhraní

interface *IExample: IUnknown* – jméno rozhraní a mateřské rozhraní, od něhož se dědí všechny jeho metody. Konvencí je stanoveno, že jméno rozhraní je ve tvaru `IInterfaceName`.

HRESULT – standardem DCOM definovaná návratová hodnota

[in, out] – typ parametru

2.9.2. Rozhraní definující properties

Properties jsou data poněkud odlišného významu, než je tomu u klasického programování. Někdy je potřebné, aby při čtení resp. zápisu proměnné byla ještě vykonána nějaká další činnost – např. při čtení

hodnoty tlakového čidla je nutné ověřit, zda teplota nepřesáhla stanovené meze a čtenou hodnotu tak lze považovat za korektní. Za tímto účelem byly zavedeny *properties.* K jejich čtení resp. zápisu se používají metody, které kromě samotného přenosu hodnoty vykonají určitou činnost. V jazyku IDL jsou za tímto účelem vyhrazena klíčová slova *propput* resp. *propget:*

```
[
  object,
  uuid(7f41e860-ba54-11d1-912c-00a02442df7d),
  oleautomation,
  dual,
  pointer_default(unique)
]
interface IReaktor: IDispatch
{
  [propput] HRESULT InflowValve( [in]          VARIANT_BOOL  newVal);
  [propget] HRESULT InflowValve( [out, retval] VARIANT_BOOL   *pVal);
  [propget] HRESULT OutflowValve([out, retval] VARIANT_BOOL   *pVal);
};
```

Příklad 2–3 IDL popis DCOM rozhraní s properties

Je nutné si uvědomit, že rozhraní je implementováno na lokálním stroji, zatímco hodnoty properties jsou fyzicky uloženy v paměti remote komponenty.

Zápis `[propput] HRESULT InflowValve([in] VARIANT_BOOL newVal)` tedy znamená:
„put value of local variable newVal to input property InflowValve of remote component“.

Zápis `[propget] HRESULT OutflowValve([out, retval] VARIANT_BOOL *pVal)` znamená:
„get value of output property OutflowValve of remote component and write it to local variable *pVal“.

Funkce s *propget* atributem musí mít stejné jméno jako čtená *property* a její poslední parametr musí být typu `[out, retval]`. Funkce s *propput* atributem musí mít stejné jméno jako zapisovaná *property* a její poslední parametr musí být typu `[in]`.

2.9.2.1. Popis input property

```
[propput] HRESULT <name> ([in]          <IDL data type>   Value);
[propget] HRESULT <name> ([out, retval] <IDL data type> *pValue);
```

„propput“ funkce zapíše hodnotu lokální proměnné Value do vstupní property „name“ remote komponenty.

„propget“ funkce čte hodnotu ze vstupní property „name“ remote komponenty a uloží ji do lokální proměnné *pValue (propget funkce nemusí být u input property definovaná, slouží pouze pro „readback“ účely)

2.9.2.2. Popis output property

```
[propget] HRESULT <name> ([out, retval] <IDL data type> *pVal);
```

„propget“ funkce čte hodnotu z výstupní property „name“ remote komponenty a uloží ji do lokální proměnné *pValue.

2.9.3. Popis CBA komponenty v IDL

```
import "oaidl.idl";
import "ocidl.idl";

//----------------------------------------------------------------------
// Interface ICBARTAuto (Standard Interface for RTAuto)
//----------------------------------------------------------------------
    [
        object,
        uuid(CBA00051-6C97-11D1-8271-00A02442DF7D),
        dual, oleautomation,
        helpstring("ICBARTAuto Interface"),
        pointer_default(unique)
    ]
    interface ICBARTAuto: IDispatch
    {
        [propget] HRESULT Name([out, retval] BSTR *pVal);

        [helpstring("Revision major.minor number of the runtime automation
object")]
            HRESULT Revision([out] SHORT *pMajor, [out] SHORT *pMinor);
    };

//-------------------------------------------------------------------------------
// Interface ICBABrowse (Standard Interface for RTAuto)
//-------------------------------------------------------------------------------
    [
        object,
        uuid(CBA00002-6C97-11D1-8271-00A02442DF7D),
        dual, oleautomation,
        helpstring("ICBABrowse Interface"),
        pointer_default(unique)
    ]
    interface ICBABrowse: IDispatch
    {
        [propget] HRESULT Count([out, retval] LONG *pVal);

        [helpstring("Browse all available items")]
            HRESULT BrowseItems(
            [in]            LONG     Offset,    // Offset - !=0 if last call did
                                                // not return all items
            [in]            LONG     MaxReturn, // At most return that many items
            [out]           VARIANT *pItem,     // SAFEARRAY(BSTR) - Itemnames
            [out, optional] VARIANT *pDataType, // SAFEARRAY(LONG) - VARTYPE
            [out, optional] VARIANT *pAccessRight); // SAFEARRAY(LONG) -
ACCESSRIGHTSDEF
    };

//-----------------------------------------------------------------------------
// Interface IPST
//-----------------------------------------------------------------------------
    [
        object,
        uuid(D1689C3C-1AF6-476a-A2E0-A0D6FC51EFEC),
        dual,
        helpstring("Profinet-server template Interface"),
        pointer_default(unique)
    ]
    interface IPST: IDispatch
    {
            [propput] HRESULT inputA(  [in]          unsigned char   Val ) ;
            [propget] HRESULT inputA(  [out, retval] unsigned char *pVal ) ;
            [propput] HRESULT inputB(  [in]          unsigned char   Val ) ;
            [propget] HRESULT inputB(  [out, retval] unsigned char *pVal ) ;

            [propget] HRESULT outputC( [out, retval] unsigned char *pVal ) ;
            [propget] HRESULT outputD( [out, retval] unsigned char *pVal ) ;
```

```
    };

//------------------------------------------------------------------------------
// IDLLib typelibrary
//------------------------------------------------------------------------------
    [
        uuid(A334AC92-BD1B-46b2-8291-E332C623D5B0),
        version(1.0),
        helpstring("idl 1.0 Typbibliothek")
    ]
    library IDLLib
    {
        importlib("stdole32.tlb");
        importlib("stdole2.tlb");

//------------------------------------------------------------------------------
// Class PST
//------------------------------------------------------------------------------

        [
            uuid(55E1CD94-BF74-44e8-8AB4-BE5796EBFD1C),
            helpstring("Profinet-server template RTAuto Class")
        ]
        coclass PST_RTAuto_Class
        {
            [default] interface IPST ;
            interface ICBARTAuto;
            interface ICBABrowse;
        };

    };
```

Příklad 2–4 IDL popis CBA komponenty

Kód, který je třeba upravit při definici nové komponenty, je zvýrazněn:

Interface ICBARTAuto: standardní popis ICBARTAuto rozhraní, ponechat v uvedeném tvaru

Interface ICBABrowse: standardní popis ICBARTAuto rozhraní, ponechat v uvedeném tvaru

Interface IPST: definice RTAuto input/output properties – změnit jméno rozhraní, vygenerovat nové guid, změnit definici input/output properties (viz kapitoly 2.9.2.1, 2.9.2.2). Pokud zařízení obsahuje více RTAuto, musí být každé RTAuto popsáno jedním rozhraním.

IDLLib Typelibrary: definice typelibrary – základní datové struktury využívané automarshallerem, viz kapitola 2.10, vygenerovat nové guid

Class PST: definice DCOM třídy, která bude dané implementovat rozhraní IPST – změnit jméno třídy, vygenerovat nové guid, uvést odkaz na RTAuto rozhraní. Pro každé RTAuto je nutné definovat jednu třídu.

2.10. Podpora IDL ve VisualC++

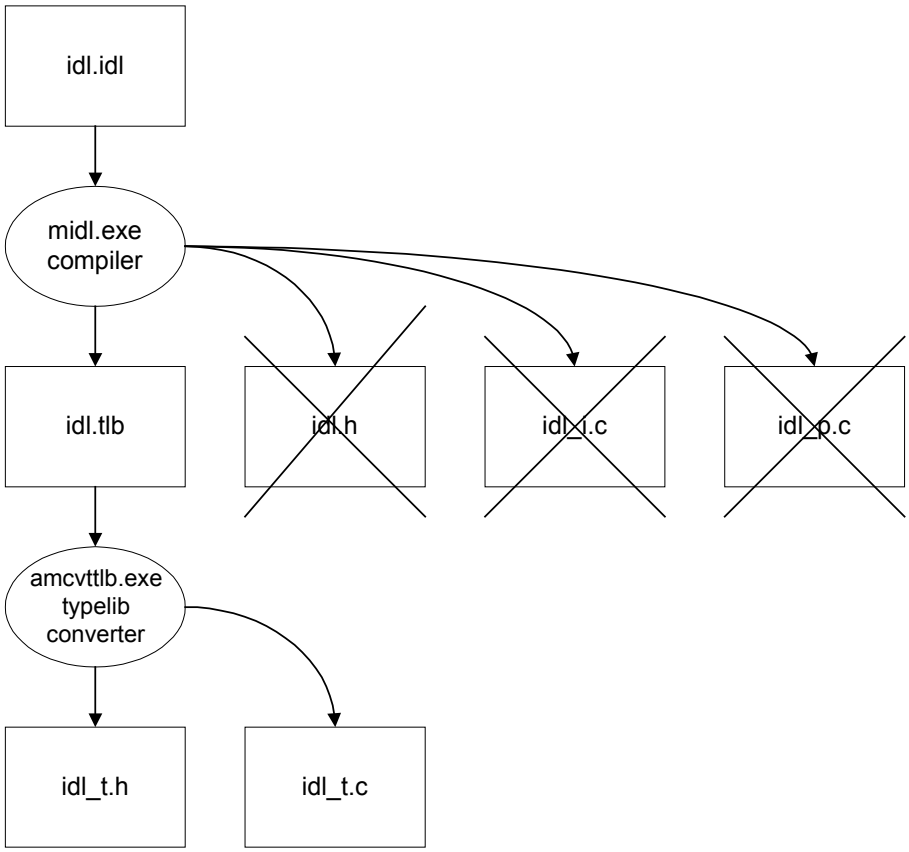
Aby bylo možné vytvořit spustitelný kód profinet–server komponenty, je třeba přeložit informace obsažené v IDL souboru do hlavičkových *.h a zdrojových *.c souborů. Pro tento účel je ve VisualC++ kompilátor IDL – `midl.exe`. Po kompilaci IDL vzniknou tyto 4 soubory:

1. **idl.tlb** – typelibrary info – binární soubor obsahující popis rozhraní, datových typů, guid kódů, vše v jediném souboru v kompaktní formě. Zjednodušeně řečeno – komprimovaná podoba souborů idl.h, idl_i.c, idl_p.c.
2. **idl.h** – deklarace třídy reprezentující rozhraní z IDL, deklarace metod reprezentující jednotlivé funkce rozhraní
3. **idl_i.c** – deklarace guid kódů
4. **idl_p.c** – kódy proxy/stub funkcí

```
IPS: public IDispatch
{
  virtual /* [propput]     */ HRESULT STDMETHODCALLTYPE put_inputA(
          /* [in]          */ unsigned char Val) = 0;
  virtual /* [propget]     */ HRESULT STDMETHODCALLTYPE get_inputB(
          /* [retval][out] */ unsigned char __RPC_FAR *pVal) = 0;
  ...
```

Příklad 2–5 idl.h s deklarací IDL rozhraní ve formě třídy

Protože profinet RS používá vlastní implementaci RPC vrstvy včetně automarshalleru, pro generování C kódu se používá následující postup, jehož základem je typelib info, tedy soubor `idl.tlb`.



Obrázek 2.19 Generování kódu pro automarhaller

Aplikace `amcvttlb.exe` je součástí RS. Z typelib info vytvoří soubory `idl_t.h, idl_t.c` s deklarací třídy `IDLLib_Tbl`, která obsahuje veškeré informace nutných pro automarshaller – seznam rozhraní,

definice jejich funkcí, seznam properties (jméno property, typ in/out, přístupová práva, datový typ). Automarshaller načte tuto třídu a má tak k dispozici veškeré informace o dané komponentě.

Postup kompilace IDL je následující:

```
>>midl.exe /Oicf idl.idl
>>amcvttlb.exe /prefix=SR idl.tlb
```

Vygenerované soubory `idl_t.h, idl_t.c` je pak nutné vložit do VisualC++ projektu. Příklad třídy `IDLLib_Tbl` viz Příloha A.

Poznámka: uvedeným postupem vygenerované soubory `idl_t.h, idl_t.c` by měly být přenositelné i na jiné platformy, oba soubory obsahují pouze deklarace konstant

2.11. Úvod do XML

XML je nový dokumentační jazyk značkovacího typu. Jeho struktura je velmi podobná HTML, ale s důrazem na datový obsah. Základními prvky XML dokumentu jsou tag, element a atribut.

2.11.1. Tag, element, atribut

Datová struktura XML souboru je tvořena stromem:

```
<?xml version="1.0" encoding=”UTF-8”? standalone="no"?>
<!DOCTYPE TEMPLATE_NAME SYSTEM "template.dtd">

<root>
  <element attribut1=“value1“ attribut2=“value2“>

    <sub-element attribut3=“value3“ attribut4=“value4“>
      content
    </sub-element>

    <sub-element>
      content
    </sub-element>

  </element>
</root>
```

Příklad 2–6 Struktura XML dokumentu

Úvodní tag `<?xml ... >` definuje verzi XML, použité kódování a umístění DTD šablony (viz následující kapitola).

Element je vymezen počátečním tagem `<element>` a koncovým tagem `</element>`. Vše mezi těmito tagy je považováno za jeho obsah. Elementy mohou být definovány 3 způsoby:

1. základní element
   `<element>content</element>`

2. element s atributem
   `<element attribut1="value1" attribut2="value2">content</element>`

3. prázdný element (bez obsahu)
   `<element attribut1="value1" attribut2="value2"/>`

Pokud chceme do XML uložit nějaká data, jsou 2 možnosti:

a) uložit data jako hodnotu elementu
b) uložit data jako atribut elementu

Sdružení W3C, které XML vyvinulo, doporučuje ukládat data jako elementy, protože to vede uživatele k tvorbě stromové struktury dokumentu, což je výhodnější z hlediska přehlednosti, navíc se takový dokument lépe zpracovává (třídění, prohledávání ...). Atribut by měl být je nositelem dodatečné informace o elementu, např.:

```
<length Unit="mm"   > 50   </length>
<width  Unit="inch" > 2.44 </width>
<deep   Unit="loket"> 1.2  </deep>
```

Příklad 2–7 XML atributy

Komentáře se uzavírají mezi tagy `<!-- comments -->`.

2.11.2. DTD šablony

Pokud uživatel požaduje, aby XML soubor měl určitou strukturu, např. obsahoval danou stromovou strukturu elementů, může toto definovat na začátku dokumentu pomocí DTD šablony (Dokument Type Definition). DTD může být do dokumentu vložen přímo jako text, nebo jako odkaz na soubor:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE TEMPLATE_NAME [

<!-- definice šablony -->

]>
```

Příklad 2–8 DTD v XML souboru

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE TEMPLATE_NAME SYSTEM "template.dtd">
```

Příklad 2–9 DTD v externím souboru

Strukturu XML lze v DTD definovat těmito prostředky:

<u>Požadujeme, aby element1 obsahoval element2, element3:</u>

```
<!ELEMENT element1 (element2+, element3+) >
```

Lze použít značek ? * + | které mají stejný význam jako v regulárních výrazech, tedy:

? žádný nebo jeden výskyt daného elementu
* žádný, jeden nebo více výskytů

+ jeden nebo více výskytů
| logická spojka OR

Požadujeme, aby element4 obsahoval atribut1, atribut2:

```
<!ATTLISTT element4 atribut1 #REQUIRED
                   atribut2 #IMPLIED>
```

Lze použít těchto značek:

`#REQUIRED`	daný atribut je požadován
`#IMPLIED`	daný atribut může ale nemusí být definován
`#FIXED "value"`	daný atribut může být vynechán – pak je vložen automaticky s hodnotou "value" ; pokud je definován, musí mít specifikovanou hodnotu jinak je vyhozena chyba

2.11.3. Popis CBA komponenty v XML

Pro popis ES komponenty je v profinet specifikaci uveden přesný popis struktury XML souboru, včetně DTD šablony. Plný výpis XML souboru viz [1] kap. 3.2.4.3, [2] kapitola 10.2, výpis DTD šablony [1] kapitola 3.4.1. Zde uvedu pouze elementy CBA XML souboru, které je třeba modifikovat ručně při definici nové komponenty typu profinet–server:

```
<!-- ESMaster -->
<coclass progid="CBAXMLServer.CBAXML" clsid="{17ec423e-5eb4-11d3-8601-
00105a705355}">
  <interface id="REF-1" iname="ICBAESMaster" iid="{4E436F12-6061-11D3-8606-
00105A705355}" enum="false">
    <import>
      <copy path=".\" targetpath=".\"/>
    </import>
    <property name="ComponentID" datatype="8">{48D270D0-A625-4a8c-BD76-
AF137F451232}</property>
    <property name="MaxObjectNameLength" datatype="3">32</property>
    <property name="ObjectName" datatype="8">PST</property>
    <property name="HelpString" datatype="8">Profinet-server template
device</property>
    <property name="Version" datatype="8">1.0</property>
    <property name="IconPath" datatype="8">pst.ico</property>
  </interface>
</coclass>
```

Obrázek 2.20 ESMaster sekce

ESMaster sekce pouze definuje pod jakým jménem bude daná komponenta vystupovat v knihovně komponent ES software. Volba jména může být v podstatě libovolná – není nijak vázána na jména ESDevice, ESAuto (resp. LDev, RTAuto), ale z důvodů přehlednosti je vhodné aby byla podobná. V uvedeném příkladu komponenta PST zastřešuje objekty PST_LDev, PST_RTAuto.

`<copy path=".\" targetpath=".\"/>`
Definuje cestu, kde jsou uloženy dodatečné soubory nutné ke konfiguraci/downloadu komponenty, např. adresář s projektem SimaticManageru.

```
<property name="ComponentID" datatype="8">{48D270D0-A625-4a8c-BD76-
AF137F451232}</property>
```
GUID identifikující danou komponentu v ES software, je nutné vygenerovat novou hodnotu pomocí guidgen.exe.

```
<property name="ObjectName" datatype="8">PST</property>
```
Jméno komponenty, pod kterým bude vystupovat v ES software, např. v knihovně komponent. Může být libovolné.

```
<property name="HelpString" datatype="8">Profinet-server template
device</property>
```
Pomocný textový popisek.

```
<property name="Version" datatype="8">1.0</property>
```
Definuje číslo verze komponenty. ES software s verzemi nijak nepracuje, mechanismus jejich číslování záleží pouze na autorovi.

```
<property name="IconPath" datatype="8">pst.ico</property>
```
Cesta k ikoně která bude reprezentovat komponentu v ES software.

```
<!-- Hyperlink Facet of the ESMaster -->
<coclass progid="CboHypex_HyperlinkFacet.CBAHypeFacet" clsid="{0F3109D4-B950-
11D4-87A8-0050DA4CC6BB}">
  <interface id="REF-5" iname="ICBAHyperlinkFacet" iid="{0F3109D3-B950-11D4-
87A8-0050DA4CC6BB}" enum="false">
    <property name="Link"
datatype="8">http://www.volny.cz/kamil.vystejn</property>
  </interface>
</coclass>
```

Obrázek 2.21 WWW odkaz

Odkaz na www stránku, kde výrobce tuto komponentu prezentuje (dokumentace, firmware, atd...).

```
<!-- ESDevice -->
<coclass progid="CBAESDeviceServer.CBAESDevice" clsid="{40B921C0-6CFC-11D3-
8618-00105A705355}">
  <interface id="REF-6" iname="ICBAESDevice" iid="{40B921BF-6CFC-11D3-8618-
00105A705355}" enum="false">
    <property name="MaxObjectNameLength" datatype="3">32</property>
    <property name="ObjectName" datatype="8">PST_LDev</property>
    <property name="HelpString" datatype="8">MyComment</property>
    <property name="Producer" datatype="8">Kamil Vystejn</property>
    <property name="Product" datatype="8">Profinet-server template
LDev</property>
    <property name="IconPath" datatype="8">pst.ico</property>
  </interface>
</coclass>
```

Obrázek 2.22 ESDevice sekce

Tato sekce popisuje ESDevice resp. LDev.

```
<property name="ObjectName" datatype="8">PST_LDev</property>
```
Definuje jméno ESDevice, pod kterým bude komponenta reprezentována v ES software. Protože ESDevice je obrazem LDev v návrhové fázi vývoje, je nutné volit stejné jméno jako LDev.

```
<property name="Producer" datatype="8">Kamil Vystejn</property>
```
Definuje výrobce komponenty.

```
<property name="Product" datatype="8">Profinet-server template LDev</property>
```
Definuje obchodní označení komponenty.

```
<property name="IconPath" datatype="8">pst.ico </property>
```
Cesta k ikoně která bude reprezentovat ESDevice ES software.

```
<!-- ESAuto -->
<coclass progid="CBAESAutoServer.CBAESAuto" clsid="{B715EB50-6F27-11D3-8619-
00105A705355}">
  <interface id="REF-20" iname="ICBAESAuto" iid="{B715EB4F-6F27-11D3-8619-
00105A705355}" enum="false">
    <property name="MaxObjectNameLength" datatype="3">32</property>
    <property name="ObjectName" datatype="8">PST_RTAuto</property>
    <property name="HelpString" datatype="8">MyComment</property>
    <property name="RTAutoCLSID" datatype="8"></property>
    <property name="IconPath" datatype="8">pst.ico</property>
    <property name="DownloadType" datatype="3">2</property>
  </interface>
</coclass>
```

Obrázek 2.23 ESAuto sekce

Tato sekce popisuje ESAuto resp RTAuto.

```
<property name="ObjectName" datatype="8">PST_RTAuto</property>
```
Definuje jméno ESAuto, pod kterým bude komponenta reprezentována v ES software. Protože ESAuto je obrazem RTAuto v návrhové fázi vývoje, je nutné volit stejné jméno jako RTAuto.

```
<property name="IconPath" datatype="8">pst.ico </property>
```
Cesta k ikoně která bude reprezentovat ESAuto v ES software.

```
<property name="DownloadType" datatype="3">2</property>
```
Definuje zda komponenta podporuje download kódu a download connections:

0	komponenta nepodporuje žádný download
1	podporuje download kódu
2	podporuje download connections
3	podporuje obojí

Tabulka 2–4 Druhy operace download

```
<!-- Connection Facet of the ESAuto-->
<coclass progid="CBAConnectionFacetServer.CBAConnectionFacet" clsid="{24EFC92A-
C294-11D3-B430-00104BA277B2}">
  <interface id="REF-28" iname="ICBAConnectionFacet" iid="{24EFC929-C294-11D3-
B430-00104BA277B2}" enum="false">
    <property name="Owner" datatype="9" pointsto="REF-20">DCOM
Object</property>
    <property name="InterfaceName" datatype="8">IPST</property>
    <property name="ConnectionIID" datatype="8">{55E1CD94-BF74-44e8-8AB4-
BE5796EBFD1C}</property>
  </interface>
</coclass>
```

Obrázek 2.24 Connection sekce

Definuje přístup k třídě implementující *ICBARTAuto* rozhraní.

```
<property name="InterfaceName" datatype="8">IPST</property>
```
Jméno RTAuto rozhraní, může být libovolné. Pokud je daná komponenta profinet–server aplikace, pak je vhodné z důvodu přehlednosti volit stejné jméno rozhraní jako v IDL souboru. V opačném případě se volí obvykle „Default–IF“.

```
<property name="ConnectionIID" datatype="8">{55E1CD94-BF74-44e8-8AB4-
BE5796EBFD1C}</property>
```
GUID třídy implementující rozhraní *InterfaceName*. Pokud je daná komponenta profinet–server aplikace, pak zadat stejné GUID, s jakým je třída definována v IDL souboru.

```
<!-- Enumerator of the Connection-Facet-->
<coclass progid="CBAConnectionFacetServer.CBAConnectors" clsid="{24EFC92C-C294-
11D3-B430-00104BA277B2}" obtainthru="REF-28" obtainproc="Connectors">
  <interface id="REF-29" iname="ICBAConnectors" iid="{24EFC92B-C294-11D3-B430-
00104BA277B2}" enum="true">
    <enumeration>
      <enummember datatype="9" pointsto="REF-30">DCOM Object</enummember>
      <enummember datatype="9" pointsto="REF-31">DCOM Object</enummember>
      <enummember datatype="9" pointsto="REF-32">DCOM Object</enummember>
      <enummember datatype="9" pointsto="REF-33">DCOM Object</enummember>
    </enumeration>
  </interface>
</coclass>
```

Obrázek 2.25 Definice počtu portů

```
<enummember datatype="9" pointsto="REF-30">DCOM Object</enummember>
```
Každý člen této sekce reprezentuje odkaz na definici portu. Hodnotu odkazu REF–xx je vhodné začít od REF–30 a postupně inkrementovat. Výše uvedený příklad tedy odkazuje na definici 4 portů. Každý port je nutné dále popsat následující sekcí:

```
<coclass progid="CBAConnectionFacetServer.CBAConnector" clsid="{24EFC928-C294-
11D3-B430-00104BA277B2}">
  <interface id="REF-30" iname="ICBAConnector" iid="{24EFC927-C294-11D3-B430-
00104BA277B2}" enum="false">
    <property name="Owner" datatype="9" pointsto="REF-28"></property>
    <property name="Name" datatype="8">inputA</property>
    <property name="HelpString" datatype="8">MyComment</property>
    <property name="Datatype" datatype="3">17</property>
    <property name="TransferType" datatype="8">{00000000-0000-0000-0000-
000000000000}</property>
    <property name="Direction" datatype="3">1</property>
  </interface>
</coclass>
```

Obrázek 2.26 Definice portu

```
<interface id="REF-30" iname="ICBAConnector" iid="{24EFC927-C294-11D3-B430-
00104BA277B2}" enum="false">
```
Definuje REF–xx identifikátor. Musí být jedinečný pro každý port.

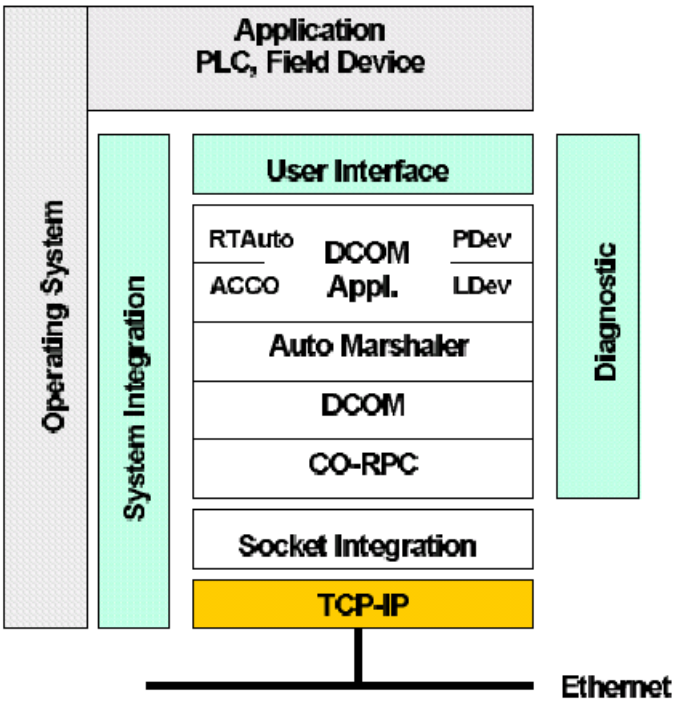
```
<property name="Name" datatype="8">InputA</property>
```
Jméno portu. Pokud je k dispozici IDL soubor, musí být jméno portu shodné s definicí odpovídající property v IDL souboru.

```
<property name="Datatype" datatype="3">17</property>
```
Datový typ portu – viz kapitola 2.7.6.

```
<property name="Direction" datatype="3">1</property>
```
Orientace portu: 1 = vstup, 2 = výstup

2.12. RS – profinet Runtime Source

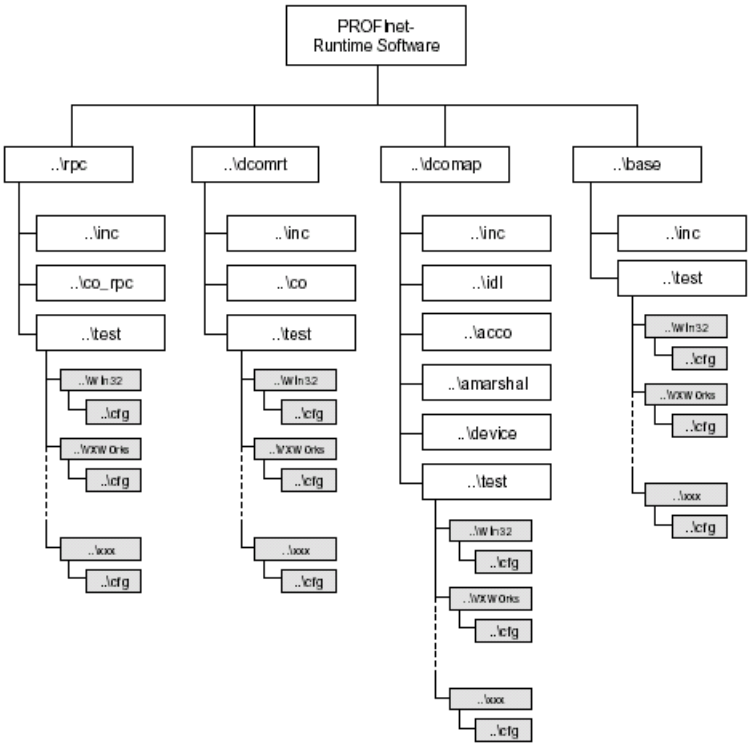
Profinet Runtime Source je balík knihoven a funkcí v programovacím jazyce C, díky němuž je možné implementovat profinet komponenty na PC se síťovým připojením.



Obrázek 2.27 Vrstvy implementované v RS

RS má vlastní implementaci DCOM, RPC a přístup k TCP/IP. Rovněž zajišťuje služby auto marshallingu.

2.12.1. Adresářová struktura RS



Obrázek 2.28 Adresářová struktura RS

- adresář base: low–level trace funkce – výpisy trasovacích informací
- adresář dcomap: přístup k RT objektům PDev, LDev, RTAuto, implementace automarsallingu, implementace ACCO
- adresář dcomrt: implementace DCOM protokolu
- adresář rpc: implementace RPC protokolu

Šedou barvou jsou označeny adresáře s kódem implementačně závislým na platformě.

2.12.2. Významné soubory RS

V dalším textu budou následující proměnné odkazovat na cestu k adresářům na přiloženém CD (viz kap. 5) – tyto proměnné pouze zastupují uvedenou cestu, nejedná se o VisualC++ projektové proměnné.

```
$LIBDIR    = CD:/lib
$SRCDIR    = CD:/src
$RSDIR     = CD:/src/rs
$TOOLSDIR  = CD:/tools
```

Veškeré zdrojové kódy RS jsou uloženy v adresáři `$RSDIR`. V prostředí VisualC++ byl založen workspace `$SRCDIR/libraries.dsw`. Přeložením vznikou následující statické knihovny:

- `$LIBDIR/Acco.lib` – implementace ACCO
- `$LIBDIR/automarhal.lib` – implemetace automarshalleru
- `$LIBDIR/Dcom.lib` – implementace DCOMu

- `$LIBDIR/Device.lib` – definice základních aplikačních tříd PDev, LDev, RTAuto a ACCO
- `$LBDIR/Rpc.lib` – implementace RPC protokolu
- `$LBDIR/Trace.lib` – implementace trasování

Z aplikačního hlediska jsou nejdůležitější tyto soubory:

Soubor	Cesta	Popis
CBARuntime_t.h **CBARuntime_t.c**	`$RSDIR/dcomap/inc/`	vygenerované hlavičky a kódy všech CBA rozhraní (viz kap. 2.5); zdrojový soubor `CBARuntime.idl` není v RS přiložen
CBAError.h	`$RSDIR/dcomap/inc/`	deklarace CBA chybových kódů, které vracejí funkce implementované v CBA rozhraních
idl.idl	`$RSDIR/dcomap/test/win32/cfg/`	IDL popis jednoduché aplikace typu profinet–server

dcomapi.h
dcomrt.c
	`$RSDIR/dcomrt/inc/`
	`$RSDIR/dcomrt/co/`
		deklarace a implementace DCOM funkcí
		`DComOsInit()`
		`DComOsOpen()`
		`DComOsClose()`

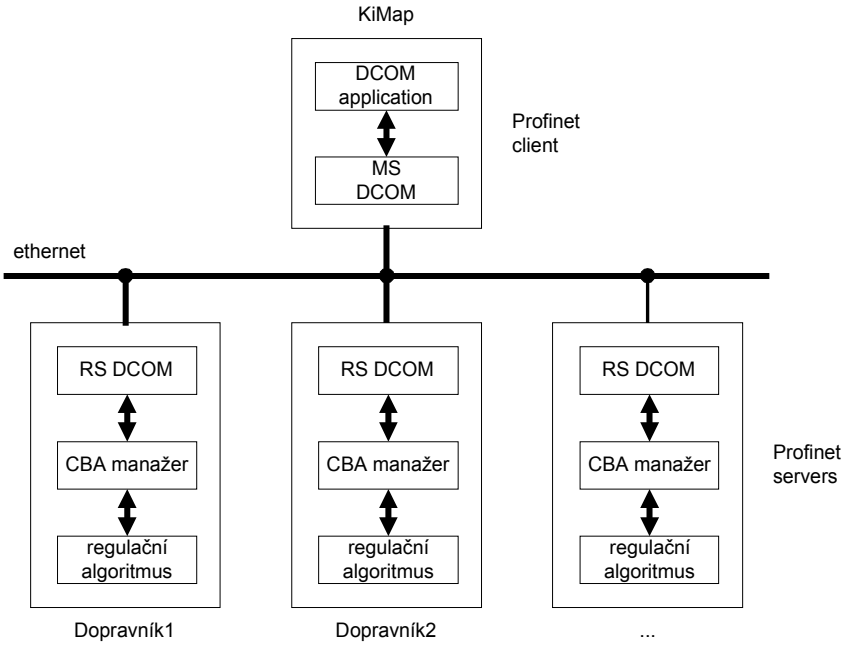
Physdev.c **Logdev.c** **RTAuto.c** **sysobj.h** **sysobj.c**	`$RSDIR/dcomap/device/` `$RSDIR/dcomap/test/`	funkce manipulující s PDev, LDev, RTAuto
TIMERS.C	`$RSDIR/dcomap/device/`	aktivace a deaktivace systémového časovače pro účely periodicky volaných funkcí (aktualizace connections, ...)
main.c	`$RSDIR/dcomap/test/win32/`	jednoduchá aplikace typu profinet–server

2.12.3. Konfigurace Windows

Z pohledu přístupu k DCOM protokolu je třeba rozlišit dva druhy aplikací:

- profinet server
- profinet klient

Každá z těchto dvou aplikací vyžaduje odlišné nastavení Windows. Důvodem je, že RS DCOM využívá port TCP/IP 135, který je standardním Microsoft DCOM obsazen. Procesy a funkce CBA manažeru (bude vysvětleno dále) vyžadují RS DCOM. Pro tyto aplikace tedy musí být MS DCOM odstraněn.
Oba materiály [1] a [2] se o problémech s MS DCOM vyjadřují velmi stručně, hlubší vysvětlení chybí.



Obrázek 2.29 Profinet–server a profinet–klient

2.12.3.1. Profinet server

Organizace PROFIbus Inc, která je autorem profinetu, vytvořila profinet RS z důvodu co nejjednodušší implementace profinetu do koncových zařízení. Nelze totiž předpokládat, že každé takové zařízení bude mít podporu DCOM&RPC zahrnutu ve svém firmware, proto RS využívá vlastní implementaci těchto protokolů. Profinet–server je aplikace, proces nebo regulační algoritmus který vykonává regulační činnost a vůči svému okolí se jeví jako CBA komponenta. Musí tedy vést vlastní správu seznamu svých LDev, RTAuto, spravovat databázi properties, aplikovat automarshaller, ACCO, poskytovat svému okolí všechna povinná rozhraní, která byla definována v kap. 2.5. Všechny tyto činnosti budeme souhrně označovat jako **CBA manažer**.

Pokud je profinet–server aplikací běžící pod Windows, pak je nutné zajistit aby DCOM&RPC implementovaný v RS nekolidoval s Microsoft DCOM&RPC instalovaným standardně ve Windows:

a) odinstalovat MS DCOM&RPC vložením následujících skriptů do Windows registru:

`$TOOLSDIR\WINDOWS_2000_dcom_off.reg` pro WINDOWS 2000 resp.
`$TOOLSDIR\WINDOWS_NT40_dcom_off.reg` pro WINDOWS NT

b) reboot

Pro zpětné nainstalování MS DCOM slouží následující skripty:

`$TOOLSDIR\WINDOWS_2000_dcom_on.reg` resp.
`$TOOLSDIR\WINDOWS_NT40_dcom_on.reg`

2.12.3.2. Profinet klient

Jedná se o aplikaci, která není CBA komponentou, ale naopak – přistupuje k těmto komponentám vzdáleně přes DCOM, jinými slovy – neběží na ní CBA manažer. Jediné, co tato aplikace musí

podporovat, je jen a pouze spolupráce s DCOM. V tomto případě je možné využít stávající MS DCOM&RPC. Pro profinet–klienta běžícího v prostředí Windows je třeba nakonfigurovat přístupová práva:

konfigurace DCOM přístupových práv:

a) spustit dcomcnfg.exe
b) na kartě Default properties povolit DCOM: Enable DCOM
c) přístupová práva (Authentication Level) nastavit na (None)
d) na kartě TCP/IP posunout protokol TCP/IP na vrchol seznamu
e) reboot

Dále je třeba doinstalovat DLL knihovnu `CBARuntimePS.dll`. Tato knihovna implementuje proxy/stub funkce pro profinet–automarshaller:

```
>> regsvr32.exe $LIBDIR/CBARuntimePS.dll
```

Poznámka: postup kompilace této knihovny je zřejmě: CBARuntime.idl (definice všech CBA rozhraní) → MIDL → CBARuntime.tlb → AmCvtTlb.exe → CBARuntime_t.h, CBARuntime_t.c → kompilace do dynamické knihovny). IDL popis CBARuntime.idl není v RS vložen, k dispozici jsou pouze zkompilované soubory .tlb, _t.h, _t.c, .dll).

DLL knihovna se odinstaluje: `>> regsvr32.exe -u CBARuntimePS.dll`

2.12.4. Profinet–server aplikace v prostředí Win32

Aplikace typu profinet–server musí splňovat následující předpoklady:

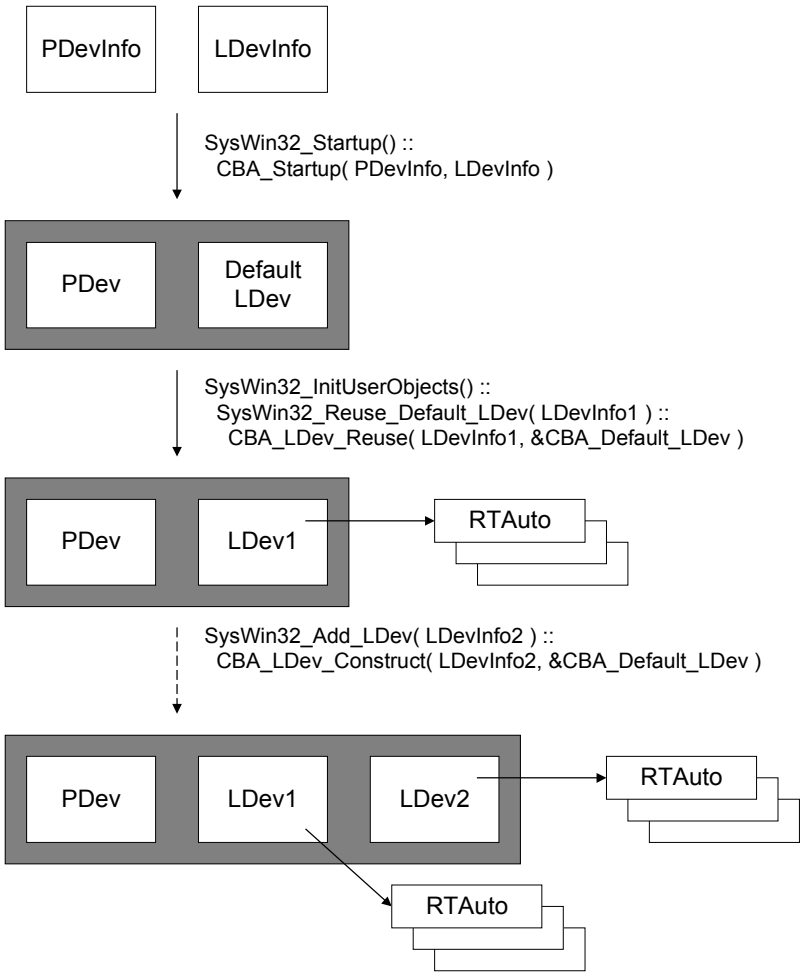
a) používat RS implementaci RPC&DCOM
b) poskytovat rozhraní *ICBAPhysicalDevice* a *ICBALogicalDevice*
c) implementovat RTAuto a poskytovat přístup k jeho properties

ad a) RS RPC&DCOM se inicializují těmito funkcemi:

```
SysWin32_DComOsInit()
SysWin32_DComOsOpen(subnet_mask, Receive_port, Send_port)
```

Profinet RS používá porty `Receive_port=Send_port=135`.

ad b), c)
Aby bylo možné nějakou komponentu používat, je nutné aby implementovala přesně jedno PDev (zajišťuje přístup k síti) a alespoň jedno LDev (abychom nad ním mohli vytvářet RTAuto zařízení). Při inicializaci profinet–serveru proto automaticky vzniká jedno PDev zařízení a jedno default–LDev zařízení. Mechanismus vzniku PDev a default–LDev demonstruje následující obrázek:



Obrázek 2.30 Vznik PDev a default–LDev

Funkce SysWin32_Startup()
Tato funkce vytváří PDev, default–LDev:

```
PDevInfo_Init( pPDevInfo );
LDevInfo_Init( pDefaultLDevInfo );
CBA_Startup(pDefaultPDevInfo, pDefaultLDevInfo )
{
  AM_Startup() ;        // inicializace automarshalleru
  CBA_Timer_Startup();  // inicializace časovačů
  CBA_Acco_Startup();   // inicializace ACCO

  // vytvoří PDev objekt
  CBA_PDev_Construct( pPDevInfo ) ;

  // vytvoří default-LDev objekt
  CBA_LDev_Construct( pDefaultLDevInfo, &CBA_Default_LDev ) ;

  CBA_LDev_Register(  CBA_Default_LDev ) ;  // registruje default-LDev u DCOM
  CBA_PDev_Register(  pPDevInfo ) ;         // registruje PDev u DCOM
}
```

Funkce SysWin32_InitUserObjects()
Tato funkce nahradí stávající default–LDev novým LDev podle definice v `sysobj.c`, vytvoří RTAuto objekty podle jejich definice v souborech `sysobj.c, idl.idl`:

```
BuildLDevInfo( pLDevInfo );
CBA_LDev_Reuse( pLDevInfo, &CBA_Default_LDev );
loop over all RTAutos of one LDev
{
  CBA_RTAuto_Construct(...);
  CBA_RTAuto_Register(...);
}
```

2.12.5. Návrh profinet–server aplikace

2.12.5.1. Definice LDev, RTAuto a properties – soubor sysobj.c

Šablona tohoto souboru je umístěna v adresáři `$SRCDIR/pst/`. V následujícím popisu jsou uvedeny pouze ty části kódu, které vhodné nebo nutné modifikovat – tyto změny jsou označeny šedým podkladem. Jako příklad je uveden popis komponenty, která implementuje jedno LDev a jedno RTAuto, protože to bude zřejmě nejčastěji využívaná konfigurace profinet–serveru.

```
/*-------------------------------------------------------------------------*/
/* Constant PDev strings for PDevInfo                                      */
/*-------------------------------------------------------------------------*/

    #define SysWin32_PDEV_PRODUCERNAME  "Kamil Vystejn"
    #define SysWin32_PDEV_PRODUCTNAME   "Profinet-server template PDev"
    #define SysWin32_PDEV_SERIALNO      "0000000001"
    #define SysWin32_PDEV_MAJOR_REV      1
    #define SysWin32_PDEV_MINOR_REV      0
```

Příklad 2–10Popis PDev

Tyto makra definují postupně: výrobce a označení komponenty, sériové číslo, číslo revize.

```
/*-------------------------------------------------------------------------*/
/* Constant LDev strings for all LDev´s                                    */
/*-------------------------------------------------------------------------*/

    #define SysWin32_LDEV_PRODUCERNAME  "Kamil Vystejn"
    #define SysWin32_LDEV_PRODUCTNAME   "Profinet-server template LDev"
    #define SysWin32_LDEV_SERIALNO      "0000000001"

    #define SysWin32_LDEV_MAJOR_REV      1
    #define SysWin32_LDEV_MINOR_REV      0
```

Příklad 2–11 Popis LDev

Makra mají stejný význam jako u PDev.

```
/*-------------------------------------------------------------------------*/
/* Number of LDevs to be constructed                                       */
/*-------------------------------------------------------------------------*/

    #define SysWin32_LDEV_COUNT     1
```

Příklad 2–12 Počet LDev zařízení

Makro definuje počet LDev zařízení.

```
/*-------------------------------------------------------------------------*/
/* Constant LDev names for all LDev´s                                      */
/* This names are used in the XML-File from engineering                    */
/* With this name the LDevs are referenced by the engineering tool         */
/*-------------------------------------------------------------------------*/

static const char * g_SysWin32_LDev_Name[SysWin32_LDEV_COUNT] =
{
    "PST_LDev"
};
```

Obrázek 2.31 Jméno LDev

Proměnná deklaruje jména jednotlivých LDev zařízení. Jméno LDev je z důvodu přehlednosti vhodné volit ve tvaru “componentname_LDev”.

```
/*-------------------------------------------------------------------------*/
/* Define the number of RTAutos for every LDev here                        */
/*-------------------------------------------------------------------------*/

static const dcom_u16 g_numRTAutosPerLDev[SysWin32_LDEV_COUNT] =
{
    1
};
```

Obrázek 2.32 Počet RTAuto pro jednotlivá LDev

Proměnná deklaruje počet RTAuto, která příslušejí pod dané LDev zařízení.

```
/*-------------------------------------------------------------------------*/
/* Define the Number of all RTAutos (build the sum over the last list)     */
/*-------------------------------------------------------------------------*/

#define SysWin32_RTAUTO_COUNT     1
```

Obrázek 2.33 Celkový počet RTAuto

Proměnná deklaruje celkový počet RTAuto v případě kdy komponenta obsahuje více než jedno LDev.

```
static const char * g_SysWin32_RTAuto_Name[SysWin32_RTAUTO_COUNT] =
{
    "PST_RTAuto"
};
```

Obrázek 2.34 Jména jednotlivých RTAuto

Proměnná deklaruje jméno RTAuto zařízení.

```
/*-------------------------------------------------------------------------*/
/* Global Data only used for the Win32 Application                         */
/*-------------------------------------------------------------------------*/

/* As pointer to the property entry you can get all informations including data
*/
SysWin32_property_t * pProperty_inputA  = NULL ;
SysWin32_property_t * pProperty_inputB  = NULL ;
SysWin32_property_t * pProperty_outputC = NULL ;
SysWin32_property_t * pProperty_outputD = NULL ;
```

Obrázek 2.35 Deklarace CBA properties

Zde je třeba deklarovat všechny property jako ukazatele na strukturu `SysWin32_property_t`. Tato struktura reprezentuje jeden záznam v databázi properties, který vede CBA manager. Ukazatele se inicializují ve funkci `SysWin32_Add_RTAuto()`:

```
/*-----------------------------------------------*/
/* Store the pointers to property table entries  */
/*-----------------------------------------------*/
if( a_usRTAutoIndex == 0 )
{
  if(strcmp(pCbaPropTbl->pName, "inputA") == 0)
    pProperty_inputA  = (SysWin32_property_t *) pCbaPropTbl->hUser;
  if(strcmp(pCbaPropTbl->pName, "inputB") == 0)
    pProperty_inputB  = (SysWin32_property_t *) pCbaPropTbl->hUser;
  if(strcmp(pCbaPropTbl->pName, "outputC") == 0)
    pProperty_outputC = (SysWin32_property_t *) pCbaPropTbl->hUser;
  if(strcmp(pCbaPropTbl->pName, "outputD") == 0)
    pProperty_outputD = (SysWin32_property_t *) pCbaPropTbl->hUser;
};
```

Obrázek 2.36 Inicializace CBA properties

Proměnná usRTAutoIndex určuje index RTAuto, lze tedy definovat rozdílné properties pro různá RTAuto. Stringy "`inputA`" , "`inputB`" atd. reprezentují jméno property tak jak jsou definována v IDL souboru.

Pokud proběhne úspěšně inicializace RTAuto objektu (voláním funkce `SysWin32_InitUserObjects()`) pak v `pProperty_inputA`, `pProperty_inputB` atd. jsou korektní ukazatele na `SysWin32_property_t` strukturu reprezentující danou property. Hodnota property je uložena na adrese, kam ukazuje člen `pPropData` této struktury. Lze tedy načíst resp. uložit hodnoty následujícím postupem:

```
    BYTE inputA  ;  // property inputA je v IDL definována jako unsigned char
    BYTE inputB  ;  // (tj. BYTE)
    BYTE outputC ;
    BYTE outputD ;

    // načtení hodnot input properties do lokálních proměnných
    inputA = *(BYTE *)pProperty_inputA->pPropData ;
    inputB = *(BYTE *)pProperty_inputB->pPropData ;

    // uložení hodnot output properties
    *(BYTE *)pProperty_outputC->pPropData = outputC ;
    *(BYTE *)pProperty_outputD->pPropData = outputD ;
```

Příklad 2–13 Manipulace s input/output properties

2.12.5.2. Hlavní program – main.c

```
//-------------------------------------------------------------------------
// inicializace trasovacího systému
//-------------------------------------------------------------------------

    #define CBA_TRACE_FILE_NAME "main.c"
    #define CBA_TRACE_FILE_ID   1
    #define CBA_TRACE_LEVEL     5
    #define CBA_COMPONENT_SYSTEM
    #include <gtrace.h>
```

```
//-----------------------------------------------------------------------------
// includes
//-----------------------------------------------------------------------------

// vložit následující hlavičkové soubory: #include “dcomcfg.h“, “dcombase.h“,
“hresult.h“, “am_cfg.h“, “am_base.h“, “automarshal.h“, “CBARuntime_t.h“,
“devcfg.h“, “interface.h“, “glist.h“, “device.h“, “accocfg.h“, “acco.h“,
“CBAError.h“, “stddev.h“, “syscfg.h“, “dcomcfg.h“, “dcomapi.h“, “trccfg.h“,
“Timers.h“

//-----------------------------------------------------------------------------
// externí funkce ze sysobj.c
//-----------------------------------------------------------------------------

    extern void                     SysWin32_InitUserObjects            (void**);
    extern void                     SysWin32_DeleteUserObjects          (void*);

    extern void                     SysWin32_All_LDev_OnStateChanged    (int);

    extern HRESULT DCOM_FKT_HUGE    SysWin32_Startup                    (void);
    extern HRESULT DCOM_FKT_HUGE    SysWin32_Shutdown                   (void);

    extern HRESULT DCOM_FKT_HUGE    SysWin32_Device_Startup             (void);
    extern HRESULT DCOM_FKT_HUGE    SysWin32_Device_Shutdown            (void);

    extern void                     SysWin32_StartTimerThread           (void);
    extern void                     SysWin32_StopTimerThread            (void);

    extern void                     SysWin32_StartPersist_Thread        (void);
    extern void                     SysWin32_StopPersist_Thread         (void);
    extern void                     SysWin32_Persist_InitConnections    (void);

//-----------------------------------------------------------------------------
// odkazy na property v registrační databázi (definice v souboru sysobj.c)
//-----------------------------------------------------------------------------

    // input properties
    extern SysWin32_property_t * pProperty_inputA ;
    extern SysWin32_property_t * pProperty_inputB ;

    // output properties
    extern SysWin32_property_t * pPropertyOutputC ;
    extern SysWin32_property_t * pPropertyOutputD ;

void main()
{
  /* inicializace DCom stacku */
  SysWin32_DComOsInit();

  /* inicializace spojení DCOM <=> RPC */
  SysWin32_DComOsOpen(“255.255.255.224“, 135, 135);

  /* inicializace CBA manageru */
  SysWin32_Device_Startup();

  /* vytvoří PDev, default LDev */
  SysWin32_Startup();

  /* interpretace typelib info idl_t.c (resp. idl.idl): */
  /*   vytvoří definovaná LDev                          */
  /*   vytvoří definovaná RTAuto                        */
  /*   vytvoří tabulku definovaných properties          */
  SysWin32_InitUserObjects(&pHandle);

  /* inicializace CBA timeru */
  SysWin32_StartTimerThread();
```

```
  /* inicializace persist threadu */
  SysWin32_StartPersist_Thread();

  /* persist BOOT - obnov přerušená connections */
  SysWin32_Persist_InitConnections();

  /* přepni všechna LDev do operačního stavu */
  SysWin32_All_LDev_OnStateChanged(CBAOperating);

  MSG msg;

  while( GetMessage (&msg, NULL, 0, 0) )
  {
    TranslateMessage( &msg );
    DispatchMessage( &msg );

    BYTE A ;
    BYTE B ;
    BYTE C ;
    BYTE D ;

    // načtení hodnot input properties do lokálních proměnných
    A = *(BYTE *)pProperty_inputA->pPropData ;
    B = *(BYTE *)pProperty_inputB->pPropData ;

    /* regulační algoritmus */
    C = f(A, B);
    D = g(A, B);

    // uložení hodnot output properties
    *(BYTE *)pProperty_outputC->pPropData = C ;
    *(BYTE *)pProperty_outputD->pPropData = D ;

    if( _kbhit() ) break ;
  }

  SysWin32_StopPersist_Thread();
  SysWin32_StopTimerThread();
  SysWin32_DeleteUserObjects(pHandle);
  SysWin32_Shutdown();
  SysWin32_Device_Shutdown();
  SysWin32_DComOsClose();
}
```

2.12.6. Trasovací systém

Během existence CBA komponenty produkuje většina spouštěných funkcí trasovací informace (traceinfo). Tyto informace jsou podrobeny filtru podle následujícího klíče:

a) úroveň `CBA_TRACE_LEVEL`: traceinfo je vypsána, pokud je její úroveň rovna nebo je menší než hodnota tohoto makra. Makro může nabývat těchto hodnot:

```
#define CBA_TRACE_LEVEL CBA_T_OFF   // = 0 trace output off
#define CBA_TRACE_LEVEL CBA_T_FATAL // = 1 fatal Error
#define CBA_TRACE_LEVEL CBA_T_ERROR // = 2 error
#define CBA_TRACE_LEVEL CBA_T_UNEXP // = 3 unexpected
#define CBA_TRACE_LEVEL CBA_T_WARN  // = 4 warning
#define CBA_TRACE_LEVEL CBA_T_NOTE  // = 5 note
#define CBA_TRACE_LEVEL CBA_T_CHAT  // = 6 all on
```

b) původ traceinfo: každá traceinfo s sebou nese zprávu o svém „původu“ (COMPONENT) tj. v kterém balíku RS (acco, dcom, ...) vznikla. Komponenta vzniku traceinfo se definuje jedním z následujících maker:

```
#define CBA_COMPONENT_UNKNOWN
#define CBA_COMPONENT_TRACE   // komponenta trace
#define CBA_COMPONENT_LPW     // ?? nedokumentováno
#define CBA_COMPONENT_RPC     // komponenta rpc
#define CBA_COMPONENT_DCOM    // komponenta dcom
#define CBA_COMPONENT_AM      // komponenta automarshaller
#define CBA_COMPONENT_DEVICE  // komponenta device
#define CBA_COMPONENT_ACCO    // komponenta acco
#define CBA_COMPONENT_S7      // ?? nedokumentováno
#define CBA_COMPONENT_SYSTEM  // uživatelská komponenta
```

V *.c souboru, kde je funkce produkující traceinfo implementována, musí být definováno právě jedno z výše uvedených maker a to buď přímo v *.c souboru, nebo nepřímo prostřednictvím hlavičkového souboru. RS používá druhou variantu – definice všech komponent vzniku traceinfo jsou umístěny v hlavičkových souborech `$RSDIR/base/inc/*trc.h`.

c) výpis souboru: definicí maker

```
#define CBA_TRACE_FILE_NAME "main.c"
#define CBA_TRACE_FILE_ID   1
```

se specifikuje soubor, v kterém je daná funkce implementována.

Samotná traceinfo se generuje funkcemi:

```
CBA_TRACE_0(level, msg)
CBA_TRACE_1(level, msg, p1)
CBA_TRACE_2(level, msg, p1, p2)
CBA_TRACE_3(level, msg, p1, p2, p3)
CBA_TRACE_4(level, msg, p1, p2, p3, p4)
```

Argument msg je string v printf formátu, p1, p2, p3, p4 jsou proměnné do stringu vkládané – je použit stejný princip jako u systémové funkce printf. Příklad:
`CBA_TRACE_2(CBA_T_ERROR, "functionX: wrong arguments values a=%d, b=%d", a, b);`

2.12.6.1. Nastavení projektu ve VisualC++

Postup návrhu profinet–server aplikace je následující:

1. vytvořit IDL popis komponenty (definovat input/output properties) viz. 2.9.3
2. definovat počet LDev, RTAuto a jejich popis (výrobce atd.) v souboru `sysobj.c` viz. 2.12.5.1
3. v souboru `sysobj.c` definovat tytéž properties jako v bodě 1. viz. 2.12.5.1
4. naprogramovat hlavní program `main.c` viz. 2.12.5.2
5. všechny tři soubory `idl.idl`, `sysobj.c`, `main.c` vložit do VisualC++ projektu (menu Project → Add To Project → Files...)
6. zajistit překlad IDL souboru MIDL kompilátorem tj. aplikovat na soubor `idl.idl` CustomBuild:

menu Project → Settings → vybrat soubor `idl.idl`
karta General: aktivovat „`Always use custom build step`“
karta Custom Build:
command: `midl.exe /Oicf $(InputPath) /out $(ProjDir)`
output: `$(ProjDir)\idl.tlb`
`$(ProjDir)\idl.h`

```
$(ProjDir)\idl_i.c
$(ProjDir)\idl_p.c
$(ProjDir)\dlldata.c
```

7. zkompilovat `idl.idl` – tak získáme soubor `idl.tlb`
8. zajistit překlad souboru `idl.tlb` typelib converterem `AMCvtTlb.exe` tj. aplikovat na soubor `idl.tlb` CustomBuild:

command: `$TOOLSDIR/AMCvtTlb.exe /prefix=SR $(InputPath)`
output: `$(ProjDir)\idl_t.h`
`$(ProjDir)\idl_t.c`

9. zkompilovat soubor `idl.tlb` – tak získáme soubor `idl_t.c`
10. soubor `idl_t.c` vložit do projektu
11. nastavit cesty k hlavičkovým souborům RS softwaru:

menu Project → Settings → Karta C/C++, položka Preprocessor, zadat cesty:

```
$RSDIR\base\inc
$RSDIR\base\test\win32\cfg
$RSDIR\dcomap\device
$RSDIR\dcomap\inc
$RSDIR\dcomap\idl\gen
$RSDIR\dcomap\test\win32
$RSDIR\dcomap\test\win32\cfg
$RSDIR\dcomrt\inc
$RSDIR\dcomrt\test\win32\cfg
$RSDIR\rpc\inc
$RSDIR\rpc\test\win32
```

12. vložit do projektu profinet knihovny:

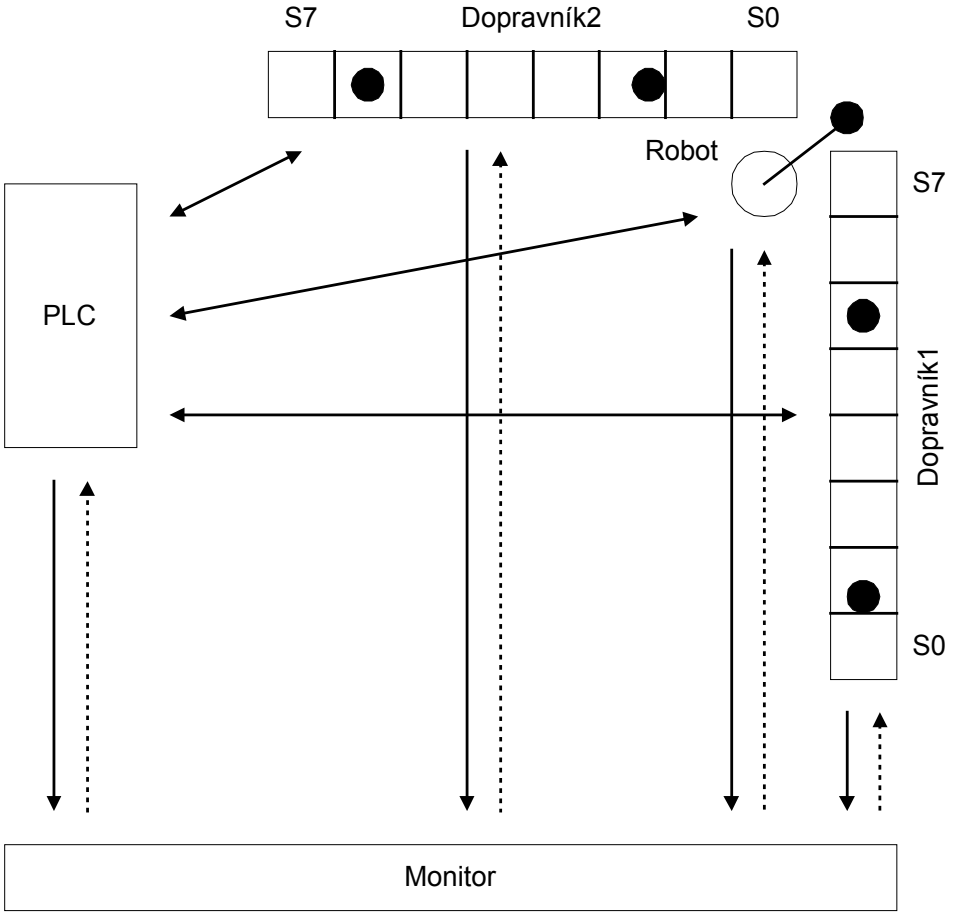
```
$LIBDIR\Acco.lib
$LIBDIR\Automarshal.lib
$LIBDIR\Dcom.lib
$LIBDIR\Device.lib
$LIBDIR\rpc.lib
$LIBDIR\Trace.lib
```

13. aplikovat Build na celý projekt

Z neznámého důvodu hází MIDL překladač chybu, pokud je nastaveno české národní prostředí – je tedy nutné ve Windows nastavit originální jazykovou verzi `English (United States)`.

3. Popis experimentálního profinet modelu

Model simuluje výrobní linku se dvěma dopravníky, robotem, PLC a monitorovacím stanovištěm. Balíky padají na Dopravník1, ten je přesouvá, na jeho konci přebírá balíky robot, který je přemisťuje na Dopravník2. Tato tři zařízení jsou řízena průmyslovým PLC, celý proces je monitorován Monitorem který vytváří vizualizační výstup. Dopravník1, Dopravník2, Robot a Monitor jsou aplikace typu profinet–server, použité PLC je CPU315–2DP s přídavnou kartu CP343–1PN, která zajišťuje profinet rozhraní a poskytuje svému okolí input/output properties. Jedná se tedy také o komponentu typu profinet–server.



Obrázek 3.1 Fyzikální schéma modelu

Pás Dopravníku1 je rozdělen do osmi sekcí S0 až S7, každá sekce je široká 25 dílků a v daný čas se v ní může nacházet nejvýše jeden balík. Pás je poháněn motorem, který je řízen pomocí pwm. Pro pwm=100% se pás pohybuje rychlostí 30 dílků za vteřinu. Pokud je nějaký balík na konci pásu v sekci S7, je nastaven flag READY, který signalizuje řídicímu algoritmu PLC, že je možné přemístit balík na Dopravník2. Dopravník1 je vybaven časovačem, který zajišťuje periodické vhazování balíků v intervalech 1–7 vteřin. Pro vizualizační účely je k dispozici informace o polohách jednotlivých balíků – výstupy P0123, P4567.

Dopravník2 je rozdělen obdobným způsobem. Pomocí flagu READY signalizuje, že je k dispozici dostatek místa pro vložení balíku (sekce S0 je volná).

Rameno robota se pohybuje v úhlové výseči 90° mezi oběma dopravníky. Robot přijímá povely motor on/off, směr vlevo/vpravo, chapadlo on/off. Na svých výstupech poskytuje pro vizualizační účely aktuální pozici (property uhel 0–90°), pro účely řízení pak indikaci o poloze – rameno v krajních dorazech (properties levy_doraz, pravy_doraz) a rameno uprostřed (property stred).

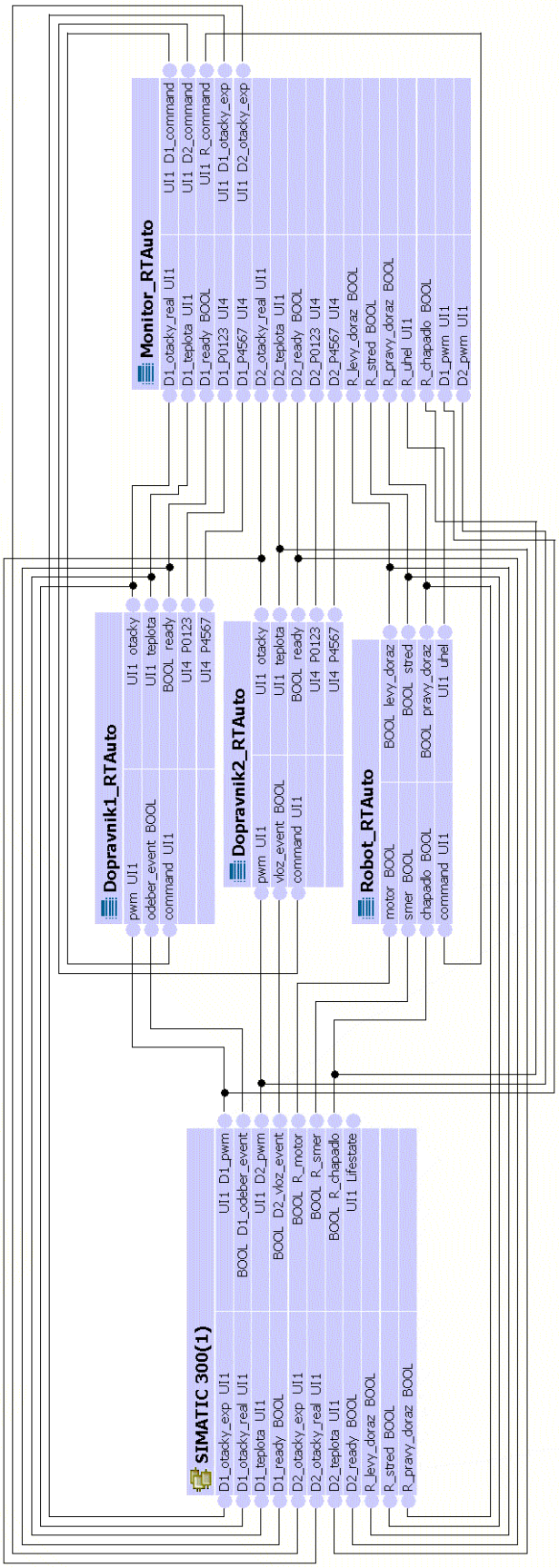
Řídicí algoritmus PLC by měl plnit následující úkoly:

a) regulovat pwm Dopravníku1 tak, aby skutečná rychlost pohybu pásu D1_otacky_real odpovídala požadované rychlosti D1_otacky_exp
b) totéž pro Dopravník2
c) zastavit pás dopravníku, pokud dojde ke zvýšení teploty motoru nad 75°C
d) přemisťovat balíky z Dopravníku1 na Dopravník2 – řídit pohyb robota

Pro účely ladění jsou k dispozici testovací módy, které se aktivují přes property command (tyto properties jsou ve výše uvedeném schématu znázorněny přerušovanou čarou):

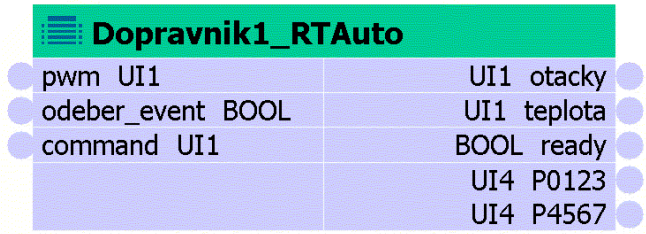
- Dopravník1 – lze nastavit periodu vhazování balíků v rozmezí 1–7 vteřin
  nastavení pevné rychlosti pohybu pásu pwm = 0, 10, 100%
  snížit/zvýšit teplotu o 1°C
  "ruční" odebrání posledního balíku
- Dopravník2 – nastavení pevné rychlosti pohybu pásu pwm = 0, 10, 100%
  snížit/zvýšit teplotu o 1°C
  "ruční" vhození balíku
- Robot – pohyb točny vlevo/vpravo
  posun točny do polohy 0°, 45°, 90°
  sevření/otevření chapadla

Akce typu „odeber balík" jsou generovány jako eventy – jako změny boolean proměnné, tj. pokud dojde ke změně „true → false" nebo „false → true" je vykonán příslušný úkon, jinak nikoli. Jména proměnných typu event budou vždy ve tvaru name_event.



Obrázek 3.2 Elektrické schéma modelu

3.1. Dopravník1



Obrázek 3.3 Dopravník1

3.1.1. Properties

pwm	pwm motoru
odeber_event	odeber z pásu poslední balík
command	aktivace testovacích módů
otacky	rychlost pohybu pásu
teplota	teplota motoru
ready	v sekci S7 je balík připraven k přesunu
P0123, P4567	polohy balíků pro účely vizualizace

Tabulka 3–1 Dopravník1 properties

pwm – pwm motoru, pro pwm=100% se pás dopravníku pohybuje rychlostí 30 dílků za vteřinu
odeber_event – pokud je vygenerován tento event, odebere se z pásu poslední balík
command – aktivuje testovací módy
otacky – otáčky motoru se mění v závislosti na pwm a počtu balíků na pásu následovně: otacky=pwm–2*bcnt (bcnt...počet balíků na pásu). Toto simuluje zatížení pásu, které vede k poklesu otáček motoru.
teplota – teplota motoru, inicializační hodnota je 25°C. Její změna se dá vyvolat testovacími signály command[5] resp. command [4].
ready – TRUE → v sekci S7 je balík (slouží jako indikace, že lze tento balík přemístit na Dopravník2)
P0123, P4567 – vizualizační výstupy, popis viz. kapitola 3.1.3

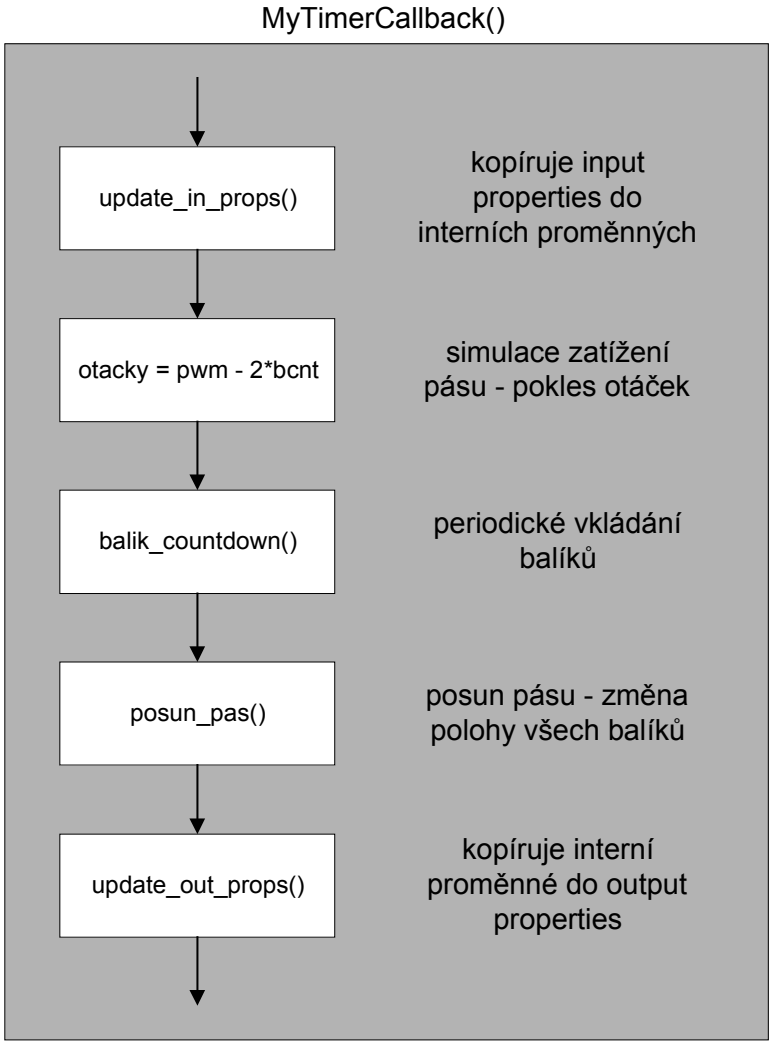
3.1.2. Testovací módy

command[7–6]	pwm motoru
00	PLC (pwm motoru řízena PLC)
01	pwm = 0%
10	pwm = 10%
11	pwm = 100%
command[5]	event: snížit teplotu o 1°C
command[4]	event: zvýšit teplotu o 1°C
command[3]	event: odeber poslední balík
command[2–0]	perioda automatického vhazování balíků [sec]

Tabulka 3–2 Dopravník1 – seznam testovacích módů

3.1.3. Popis programu

V programu je nadefinován časovač s periodou 10ms, jehož Callback funkce periodicky vykonává všechny potřebné operace tj. aktualizuje properties, zajišťuje regulační algoritmus atd, viz následující obrázek:



Obrázek 3.4 Dopravník1 – timer callback funkce

Celkový počet balíků na pásu je uložen v proměnné **bcnt**, jejich polohu na pásu udává pole **P[0–7]**. Aby nedocházelo k nepotřebnému zatěžování sítě pro přenos polohy všech balíků směrem k vizualizačnímu nástroji, je poloha balíků sloučena do dvou výstupních properties P0123, P4567 typu unsigned long int s následujícím mapováním:

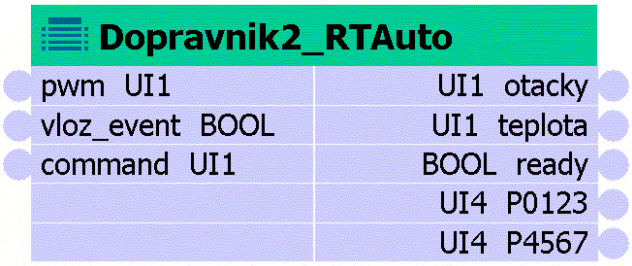
P0123[bit 0–7] = P[0]: poloha balíku č.0
P0123[bit 8–15] = P[1]: poloha balíku č.1
P0123[bit 16–23] = P[2]: poloha balíku č.2
P0123[bit 24–31] = P[3]: poloha balíku č.3

P4567[bit 0–7] = P[4]: poloha balíku č.4
P4567[bit 8–15] = P[5]: poloha balíku č.5
P4567[bit 16–23] = P[6]: poloha balíku č.6
P4567[bit 24–31] = P[7]: poloha balíku č.7

Pokud jsou například na pásu 3 balíky v poloze 10, 50, 90 dílků, pak bcnt=3, P[0]=10, P[1]=50, P[2]=90. Protože na pásu jsou pouze 3 balíky, tak poziční proměnné P[3-7] neobsahují relevantní údaj,

proto jsou nastaveny na hodnotu 255. Tímto principem je zajištěno, že vizualizační nástroj (Monitor) bezpečně pozná počet balíků na pásu pouze z properties P0123, P4567. Není tedy třeba přenášet proměnnou bcnt.

3.2. Dopravník2



Obrázek 3.5 Dopravník2

3.2.1. Properties

pwm	pwm motoru
vloz_event	vložení balíku
command	aktivace testovacích módů
otacky	rychlost pohybu pásu
teplota	teplota motoru
ready	sekce S0 je volná – je možné vložit balík
P0123, P4567	polohy balíků pro účely vizualizace

Tabulka 3–3 Dopravník2 properties

vloz_event – pokud je vygenerován tento event, vloží se na pás nový balík
ready – TRUE → v sekci S0 se nenachází žádný balík (lze tedy bezpečně vložit nový)

Ostatní properties mají stejný význam jako u Dopravníku1.

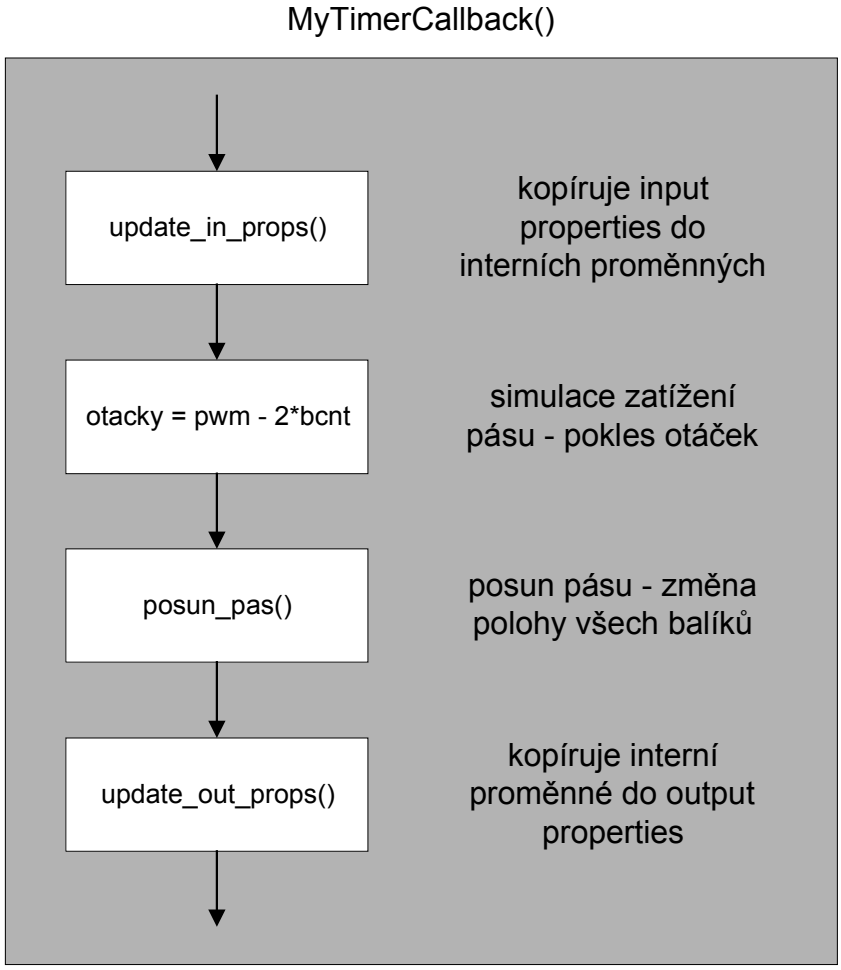
3.2.2. Testovací módy

command[4–3]	pwm motoru
00	PLC (pwm motoru řízena PLC)
01	pwm = 0
10	pwm = 10
11	pwm = 100
command[2]	event: snížit teplotu o 1°C
command[1]	event: zvýšit teplotu o 1°C
command[0]	event: vlož balík

Tabulka 3–4 Dopravník2 – seznam testovacích módů

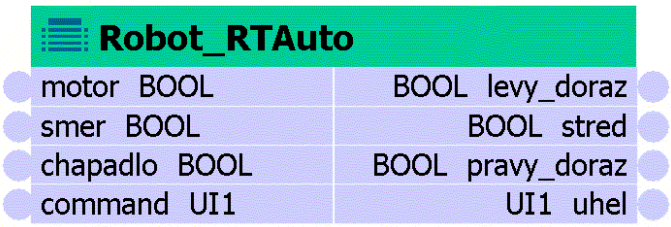
3.2.3. Popis programu

Je použit časovač 10ms podobně jako u Dopravníku1:



Obrázek 3.6 Dopravník2 – timer callback funkce

3.3. Robot



Obrázek 3.7 Robot

3.3.1. Properties

motor	motor on/off
smer	směr pohybu točny (vlevo=true)
chapadlo	true = sevřené chapadlo
command	aktivace testovacích módů
levy_doraz	točna v poloze 90°
stred	točna v poloze 45°
pravy_doraz	točna v poloze 90°
uhel	poloha točny

Tabulka 3–5 Robot properties

motor – pokud je true, točna se otáčí směrem určeným pomocí property **smer**
smer – TRUE ... pohyb vlevo (proti směru hodinových ručiček)
– FALSE ... pohyb vpravo (po směru hodinových ručiček)
chapadlo – TRUE ... chapadlo sevřené
command – aktivace testovacích módů
uhel – poloha točny ve stupních
levy_doraz – točna se nachází v poloze 90°
stred – točna se nachází v poloze 45°
pravy_doraz – točna se nachází v poloze 0°

3.3.2. Testovací módy

command[4–3]	chapadlo
00	PLC (chapadlo řízeno PLC)
01	otevři chapadlo
10	sevři chapadlo
command[2–0]	pohyb točny
000	PLC (točna řízena PLC)
001	„don't move“ – motor off
010	„move left“ – motor on, směr vlevo
011	„move right“ – motor on, směr vpravo
100	„move to 0 degrees“ – uhel = 0°

101	„move to 45 degrees“ – uhel = 45°
110	„move to 90 degrees“ – uhel = 90°

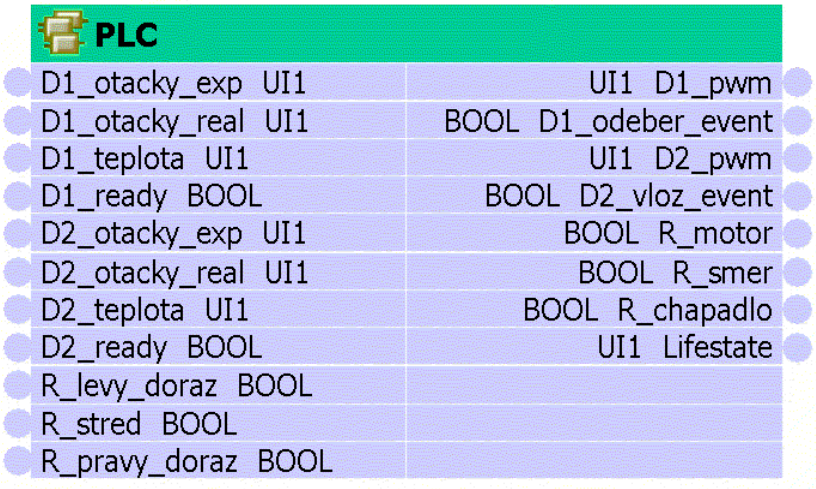
Tabulka 3–6 Robot – seznam testovacích módů

3.3.3. Popis programu

```
VOID CALLBACK MyTimerCallback(...)
{
    update_in_props() ;

    if( motor )
        if( smer )  // vlevo
            uhel += SPEED*((float)0.001*MYTIMER_PERIOD) ;
        else
            uhel -= SPEED*((float)0.001*MYTIMER_PERIOD) ;

    // limits
    if( uhel < 0  ) uhel = 0  ;
    if( uhel > 90 ) uhel = 90 ;

    // update flagu
    pravy_doraz = ( uhel <= 0 ) ;
    stred       = ( uhel >= 42 && uhel <= 48 ) ;
    levy_doraz  = ( uhel >= 90 ) ;

    update_out_props();
}
```

Obrázek 3.8 Robot – timer callback funkce

3.4. PLC



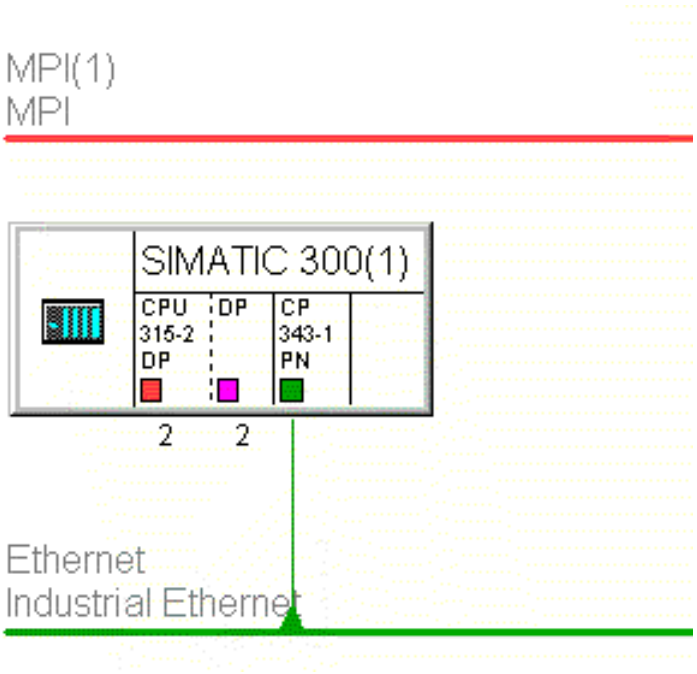
Obrázek 3.9 PLC

3.4.1. Properties

D1_otacky_exp	Dopravník1: požadované otáčky
D1_otacky_real	Dopravník1: skutečné otáčky
D1_teplota	Dopravník1: teplota
D1_ready	Dopravník1: balík v sekci S7
D2_otacky_exp	Dopravník2: požadované otáčky
D2_otacky_real	Dopravník2: skutečné otáčky
D2_teplota	Dopravník2: teplota
D2_ready	Dopravník2: S0 je volná
R_levy_doraz	Robot: točna v poloze 90°
R_stred	Robot: točna v poloze 45°
R_pravy_doraz	Robot: točna v poloze 0°
D1_pwm	Dopravník1: pwm
D1_odeber_event	Dopravník1: odeber poslední balík
D2_pwm	Dopravník2: pwm
D2_vloz_event	Dopravník2: vlož balík
R_motor	Robot: pohyb točny
R_smer	Robot: směr pohybu
R_chapadlo	Robot: otevření/sevření chapadla
Lifestate	operační stav PLC LDev

Tabulka 3–7 PLC properties

3.4.2. Hardwarová konfigurace



Obrázek 3.10 PLC – hardwarová konfigurace

Konfigurace sestává z PLC CPU315–2 DP a přídavné karty CP343–1PN, která zajišťuje přístup k ethernetu a simuluje profinet rozhraní. Pro korektní přístup na ethernet je třeba nastavit mapování PG/PC interface následovně:

Control Panel → PG/PC Interface: CP_PN_1 → TCP/IP, S7ONLINE → TCP/IP

Po dodání karty CP343–1PN není od výrobce standardně nastavena IP adresa – to je třeba provést ručně v hardwarové konfiguraci Simatic Manageru – kliknout pravým tlačítkem myši na CP343–1PN → Object properties → karta General → Interface → Properties. Poté provést download konfigurace přes konfigurační rozhraní MPI. Po natažení konfigurace je PLC přístupné přes ethernet na specifikované IP adrese. Přes ethernet je pak možné provádět download, monitoring v Simatic Manageru a přistupovat k profinet rozhraní, pokud je v projektu definováno funkčním blokem FB88 resp. datovým blokem DB100 – viz následující kapitola.

3.4.3. Mechanismus přenosu properties

Podporu profinet properties zajišťuje funkční blok FB88 („PN_InOut“) a globální datový blok, který bude v souladu s [3] označován DB100 („PN_Interface_DB“). V tomto datovém bloku se specifikují input/output properties následujícícm postupem:

a) DB100 musí mít definován atribut `CBA_db = true`
a) každá datová položka v DB100 specifikuje jednu property
b) je třeba zadávat property v pořadí: nejdříve všechny vstupní properties, pak všechny výstupní properties
c) paměť alokovaná pro vstupní properties musí být velikosti násobku WORD, jinak řečeno – první output property musí ležet na sudé adrese; pokud toto není splněno, je třeba vložit dummy

input property a tím zaokrouhlit velikost paměti alokované pro input properties na násobek WORD

d) první input property musí mít definovány atributy:

```
CBA_direction = in
CBA_transfer_type = connectable
```

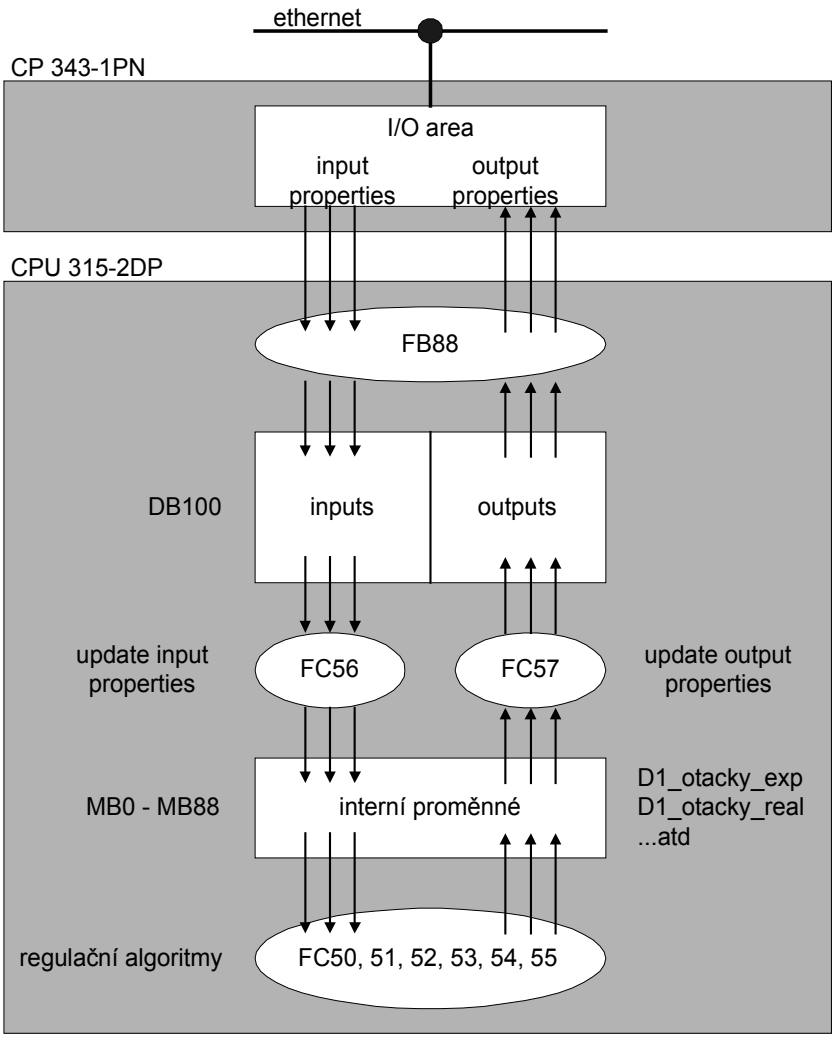
e) první output property musí mít definován atribut:

```
CBA_direction = out
```

Poznámka: atributy se nastavují následovně: kliknout na daný objekt, přes pravé tlačítko myši vybrat Object properties, pak vybrat kartu Attributes

Příklad datového bloku DB100 viz Příloha G. Znak vlaječky označuje položky s atributy.

Pro přístup k properties z řídicího algoritmu lze jednoduše použít tečkové konvence, např. `L "PN_Interface_DB".D1_otacky_exp,` což ale není příliš přehledné, proto jsou pro všechny properties vytvořeny v paměti CPU jejich obrazy ve formě interních proměnných (např. D1_otacky_exp = MB0). Řídicí algoritmus pracuje s těmito proměnnými, přičemž musí být zajištěna jejich periodická aktualizace a to pro input i output properties. K tomuto účelu jsou naprogramovány funkční bloky FC56, FC57. Koncepce práce s properties je znázorněna na následujícím obrázku:



Obrázek 3.11 PLC – mechanismus přenosu properties

3.4.4. Seznam interních proměnných

Kopie vstupních a výstupních properties:
D1_otacky_exp, D1_otacky_real, D1_teplota, D1_ready
D2_otacky_exp, D2_otacky_real, D2_teplota, D2_ready
R_levy_doraz, R_stred, R_pravy_doraz
D1_pwm, D1_odeber_event
D2_pwm, D2_vloz_event
R_motor, R_smer

Popis těchto proměnných není třeba opakovat, viz kapitola 3.4.1.

D1_run – hodnotou FALSE se indikuje přehřátí motoru Dopravníku1 (D1_teplota > 75°C)
D2_run – totéž pro Dopravník2

R_stav – aktuální stav stavového automatu řízení robota ve formě BYTE proměnné, kódování viz Tabulka 3–8

FB88_DONE, FB88_ERROR, FB88_STATUS – stavové proměnné bloku FB88 (popis viz [4] kapitola 5.2)

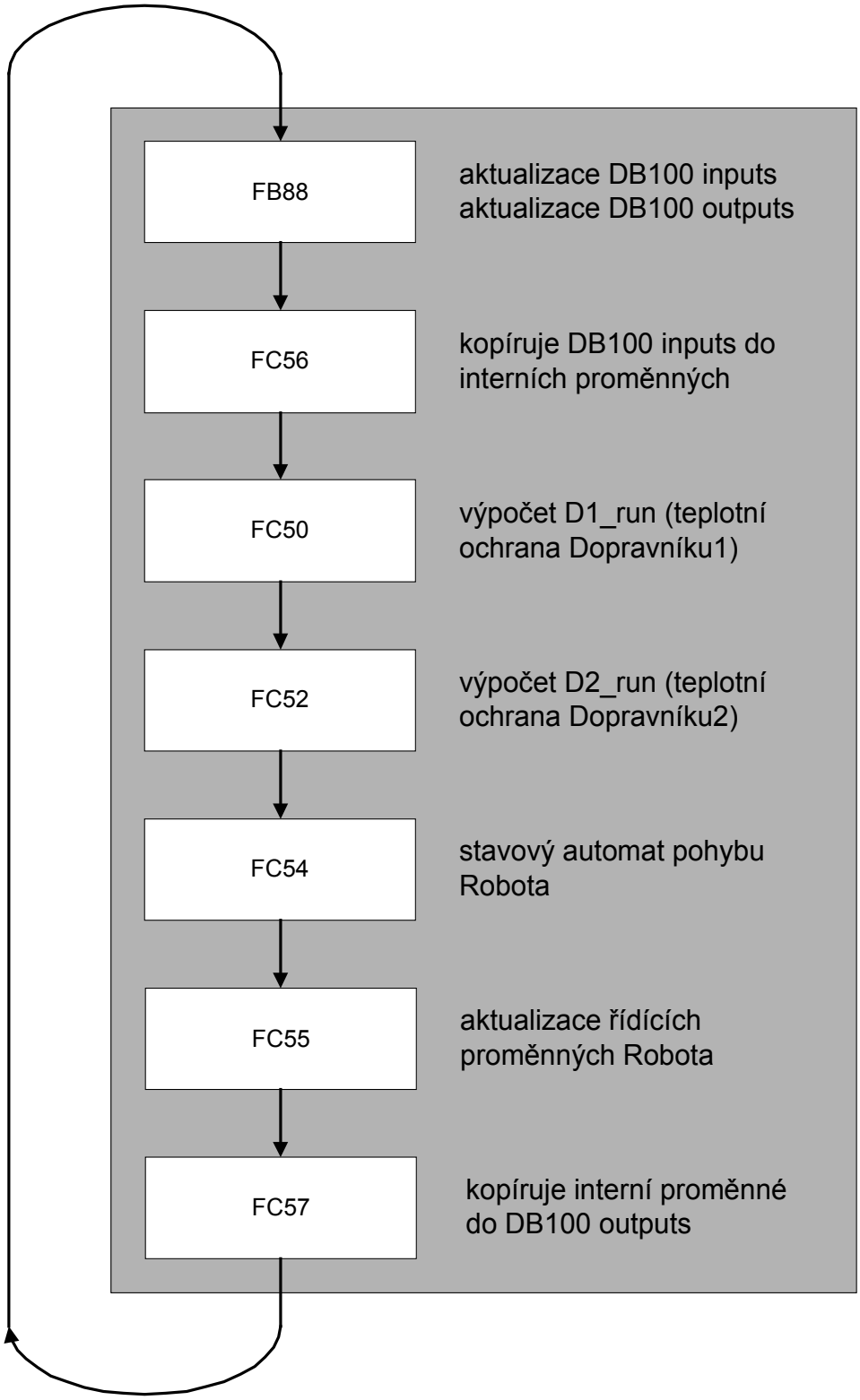
Umístění jednotlivých proměnných v paměti lze nalézt v příloze Příloha J.

3.4.5. Popis programu

- **OB1** – hlavní scan cycklus, aktualizuje input/output properties, spouští teplotní ochranu motorů Dopravníku1,2, spouští stavový automat řízení pohybu robota
- **OB35** – s periodou 100ms provádí akční zásahy nad D1_pwm, D2_pwm
- **FC50** – monitoruje teplotu Dopravníku1 a v případě přehřátí zablokuje regulační zásah nad D1_pwm
- **FC51** – výpočet akčního zásahu nad D1_pwm
- **FC52, 53** – totéž pro Dopravník2
- **FC54, 55** – stavový automat pohybu robota
- **FC56,57** – aktualizace DB100 ⇔ interní proměnné

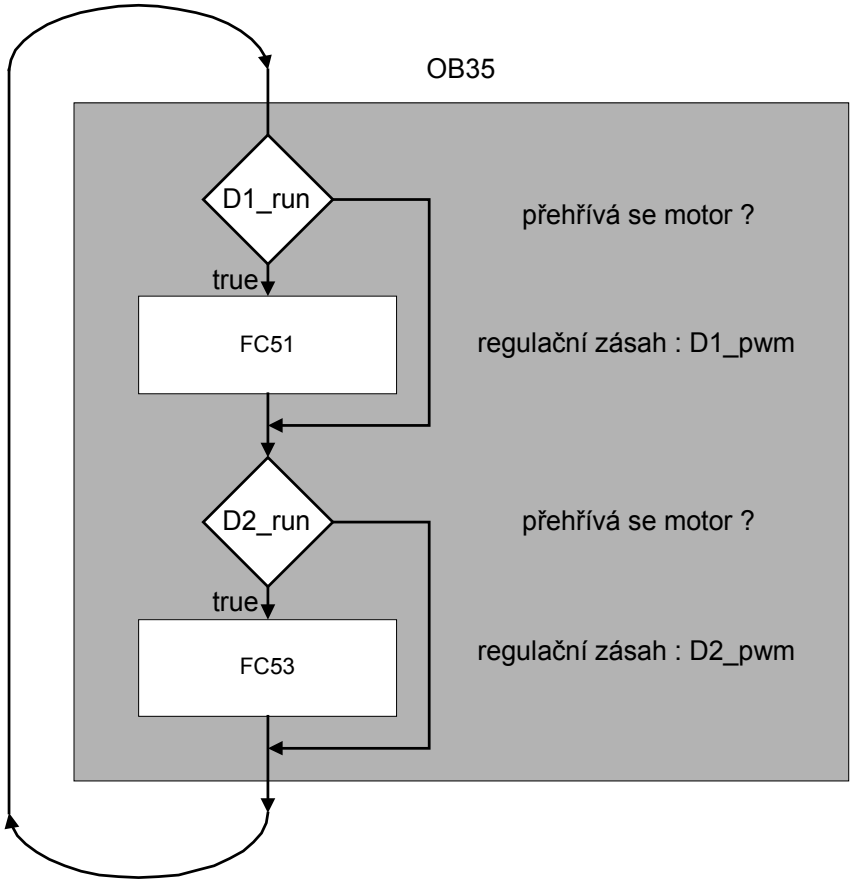
Důvodem, proč není výpočet akčního zásahu prováděn v hlavním scan cyklu je požadavek na synchronizaci akčního zásahu s periodou QoS přenosu dat mezi PLC a dopravníky. Vzhledem k demonstračnímu účelu experimentálního modelu byla použita triviální metoda výpočtu – inkrementace/dekrementace pwm v závislosti na vztahu D1_otacky_real <> D1_otacky_exp. Scan cyklus PLC je 8ms, zatímco QoS=100ms. To znamená, že během jednoho datového transferu properties dopravník ↔ PLC se OB1 vykoná přibližně 12x. Pokud by tedy byl akční zásah prováděn v hlavním scan cyklu, nestačil by dopravník reagovat na rychlé změny pwm, což by vedlo k oscilacím pwm. Uvedená metoda výpočtu akčního zásahu je funkční pouze za podmínky, že výpočet akčního zásahu se provádí se stejnou nebo větší periodou než QoS. Vzhledem k tomu, že QoS transferu dat je nastaven na 100ms, byla tato hodnota použita i pro periodu volání funkcí realizujících akční zásah v OB35.

3.4.6. OB1: hlavní scan cyklus



Obrázek 3.12 PLC – OB1: hlavní scan cyklus

3.4.7. OB35: periodický scan cyklus 100ms



Obrázek 3.13 PLC – OB35: periodický cyklus 100ms

Proměnná D1_run ve stavu true indikuje, že teplota motoru je v korektních mezích (<75°C). Jakmile se motor přehřeje, nastaví se D1_run do stavu false. V tom případě se zablokuje algoritmus regulačního zásahu a zároveň se zabrání přenosu hodnoty D1_pwm do výstupní property DB100.D1_pwm:

```
FC57: if( D1_run )
  DB100.D1_pwm = D1_pwm ;  // OK
else
  DB100.D1_pwm = 0 ;  // přehřátí motoru
end if;
```

Totéž platí pro Dopravník2.

3.4.8. FC50: teplotní ochrana Dopravníku1

```
// zastav pás, pokud se motor zahřívá
if( D1_teplota > 75 )
  D1_run = FALSE ;
else
  D1_run = TRUE ;
end if;
```

3.4.9. FC51: regulační algoritmus Dopravníku1

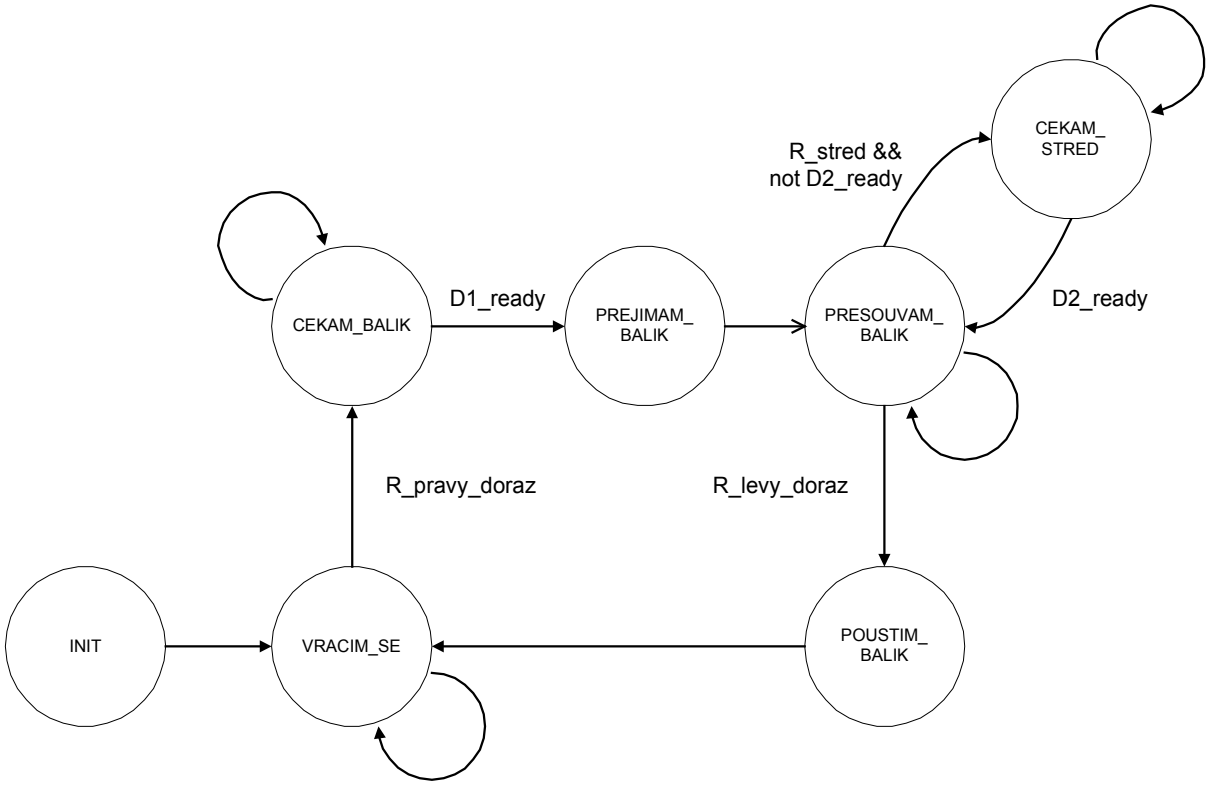
```
// regulační zásah

if( D1_otacky_real > D1_otacky_exp ) D1_pwm-- ;
if( D1_otacky_real < D1_otacky_exp ) D1_pwm++ ;


// ochrana podtečení/přetečení
if( D1_pwm == 255 ) D1_pwm = 0 ;
if( D1_pwm > 100 ) D1_pwm = 100 ;
```

3.4.10. FC52, FC53: teplotní ochrana, regulační algoritmus Dopravníku2

Je použit tentýž postup jako u FC50 resp. FC51.

3.4.11. FC54: regulační algoritmus pohybu robota

Algoritmus je popsán následujícím stavovým diagramem:



Obrázek 3.14 FC54: stavový diagram pohybu robota

Stavový diagram je naprogramován ve tvaru podmínek if–then–else v jazyce STL. Aktuální stav se uchovává v proměnné R_stav. Výstupní proměnné stavového automatu (R_motor, R_smer, R_chapadlo, D1_odeber_event, D2_vloz_event) se kódují v FC55. INIT je pouze pseudostav demonstrující přechod do VRACIM_SE po resetu PLC.

Ve stavu CEKAM_BALIK je točna robota v poloze 0°, tedy chapadlem přesně nad sekcí S7 Dopravníku1. V této poloze se čeká na příjezd balíku. Jakmile balík přijede (indikováno vstupem D1_ready), vygeneruje se event D1_odeber_event (přechodem do stavu PREJIMAM_BALIK) – tímto se dává Dopravníku1 na vědomí, aby z pole P[0-7] odebral poslední balík. Poté se přejde do stavu

PRESOUVAM_BALIK, během kterého robot uchopí chapadlem balík a přesouvá točnu proti směru hodinových ručiček směrem k Dopravníku2. Jakmile dorazí do polohy 45° (detekováno vstupem R_stred) otestuje se, zdali je sekce S0 Dopravníku2 volná (detekováno vstupem D2_ready) – pokud ne, setrvává robot v této poloze, dokud se S0 neuvolní (stav CEKAM_STRED). Jakmile toto nastane, pokračuje se v pohybu. Když dorazí chapadlo nad sekci S0 Dopravníku2 (detekováno vstupem R_levy_doraz), pustí balík a informuje o tom Dopravník2 (pomocí eventu D2_vloz_balik – generováno ve stavu POUSTIM_BALIK). Pak se robot vrací (stav VRACIM_SE) po směru hodinových ručiček do výchozí polohy 0° (detekováno vstupem R_pravy_doraz). Poté se celý cyklus opakuje.

stav	R_stav	motor	chapadlo	generované events
INIT	–	–	–	–
CEKAM_BALIK	0	off	off	
PREJIMAM_BALIK	1	off	ON	D1_odeber_event
PRESOUVAM_BALIK	2	VLEVO	ON	
CEKAM_STRED	3	off	ON	
POUSTIM_BALIK	4	off	off	D2_vloz_event
VRACIM_SE	5	VPRAVO	off	

Tabulka 3–8 FC55: kódování stavů robota

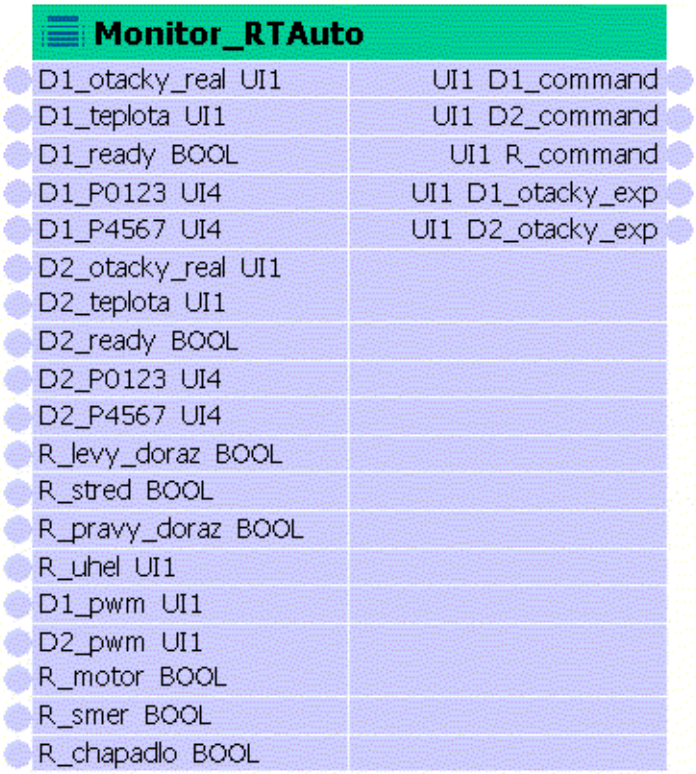
3.4.12. Export XML souboru

XML popis se v Simatic Mangeru exportuje příkazem Edit → Create Profinet component. V následujícím dialogu zadat jméno komponenty (např. PLC), zvolit vygenerování nového GUID kódu (identifications GUIDs → new) a zadat cestu k uložení XML (karta Storages area → zvolit File system a zadat cestu). Simatic Manager při generování XML vytvoří adresář, ve kterém kromě samotného XML souboru bude uložen i kompletní S7 projekt.

3.4.13. Download kódu v iMapu

1. Nejdříve je třeba založit novu knihovnu: Library → New...
2. Do knihovny importujeme XML soubor: Library → Import Components
3. Založíme nový projekt: Project → New
4. Do projektu vložíme komponentu přetažením myší z Library window do Plant View
5. Přepnout zobrazení na Network view a definovat IP adresu komponenty a masku podsítě: označit komponentu myší a přes pravé tlačítko vybrat Properties.
6. Spustit fázi „generate“ (obdoba kompilace): Project → Generate → All
7. Provést download: označit komponentu myší a přes pravé tlačítko vybrat Download Selected Device → Program only

3.5. Monitor



Obrázek 3.15 Monitor

3.5.1. Properties

Všechny in/out properties jsou vedeny z komponent Dopravník1, Dopravník2, Robot a PLC, jejich popis viz. kapitoly 3.1.1, 3.2.1, 3.3.1, 3.4.1.

3.5.2. Popis programu

```
switch (message)
{
    case WM_TIMER:

        // kopíruje input properties do interních proměnných
        update_in_props();

        // překreslí okno
        InvalidateRect(hWnd, &rectZone, TRUE);

        // kopíruje interní proměnné do output properties
        update_out_props();

        break;

    case WM_PAINT:

        paint_scene(hWnd);  // překreslí okno
        break;

    case WM_COMMAND:
```

```
switch (LOWORD(wParam))
{
    // obsluha klávesových zkratek

    case IDA_D1_otacky_exp_minus:
         if( D1_otacky_exp > 0 ) D1_otacky_exp-- ;
         break;
    case IDA_D1_otacky_exp_plus:
         ...
}
```

Protože hlavním úkolem Monitoru je vytvářet vizualizační výstup, bylo nutné napsat vlastní obsluhu fronty zpráv, která odchytává tyto:

a) WM_TIMER – nastalo přerušení časovače
b) WM_PAINT – požadavek na překreslení okna
c) WM_acc – zprávy reagující na stisk horkých kláves
d) WM_CLOSE, WM_DESTROY – zavření (ukončení) aplikace

Na rozdíl od aplikací Dopravník1,2 a Robot je obsluha časovače implementována jako reakce na zprávu `WM_TIMER`. Při příchodu této zprávy se provede update input/output properties a zašle se zpráva `WM_PAINT`, která představuje požadavek na překreslení okna. Vizuální výstup je tedy aktualizován synchronně se změnou properties.

Při příchodu zprávy `WM_PAINT` se volá funkce paint_scene(), která nejdříve smaže obsah okna a pak jej kompletně překreslí.

Zprávy `WM_acc` jsou zasílány při stisku nadefinovaných horkých kláves – toho je využito pro aktivaci testovacích módů. Seznam klávesových zkratek viz. Příloha E.

Princip práce s properties (update input/output properties) zůstává stejný jako u již popsaných aplikací. Většinu zdrojového kódu Monitoru tvoří práce s grafikou, což zřejmě nemá význam podrobněji popisovat. Zde se spokojíme pouze s odkazem na výsledný zjev – viz. Příloha D. uveden v.

4. KiMap

KiMap je aplikace typu profinet–klient, která supluje download funkci iMapu. Umí načítat XML popis komponent, definovat datová spojení a provádět jejich download. Podporuje datová spojení těchto typů: VT_UI1, VT_UI2, VT_UI4, VT_I1, VT_I2, VT_I4, VT_R4, VT_R8, VT_BOOL.

4.1. Implementace EXPAT XML Parser knihovny

Knihovna EXPAT je OpenSource XML parser napsaný v jazyce C++. Rozklad XML souboru se provádí následujícím postupem:

```
XML_Parser xml_parser = XML_ParserCreate(NULL);

XML_SetStartElementHandler( xml_parser, StartElementHandler);
XML_SetDefaultHandler(      xml_parser, DefaultHandler);
XML_SetEndElementHandler(   xml_parser, EndElementHandler);
XML_SetUserData(            xml_parser, (void *) &cnode);

do
{
  char buff[BUFF_SIZE];

  int len = fread(buff, 1, sizeof(buff), xmlfid);
  XML_Parse(xml_parser, buff, len, feof(xmlfid));
}
while( !feof(xmlfid) );

XML_ParserFree(xml_parser);
```

Příklad 4–1 Rozklad XML souboru

Nejdříve se vytvoří instance parseru v paměti. Poté se definuje:

1. funkce, která bude vyvolána jakmile parser načte start element tag: StartElementHandler
2. funkce, která bude vyvolána jakmile parser načte obsah elementu: DefaultElementHandler
3. funkce, která bude vyvolána jakmile parser načte end element tag: EndElementHandler
4. uživatelská data (tato data jsou předávaná každé parserem volané funkci)

V následující smyčce se zpracovává XML soubor po blocích velikosti `BUFF_SIZE` bytů. Jakmile XML parser narazí na start element tag, obsah elementu nebo end element tag, vyvolá příslušnou funkci. Tato funkce obdrží tyto argumenty:

```
StartElementHandler(void *userData, const XML_Char *name, const XML_Char **atts);
```

- **userData** uživatelská data
- **name** jméno elementu
- **atts** pole atributů ve formě pole:

```
atts[0]=attribut0_name
atts[1]=attribut0_value
atts[2]=attribut1_name
atts[3]=attribut1_value

, ...,
atts[x]=“““ (konec pole)
```

```
DefaultHandler(void *userData, const XML_Char *s, int len);
```

- **userData** uživatelská data
- **s** obsah elementu
- **len** počet platných znaků v řetězci s

```
EndElementHandler(void *userData, const XML_Char *name);
```

- **userData** uživatelská data
- **name** jméno elementu

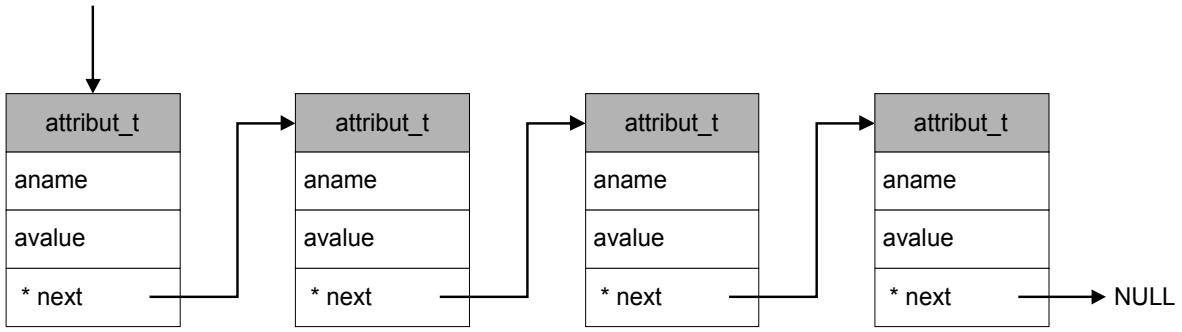
Jak budou dané funkce s danými argumenty zacházet, záleží na jejich implementaci.

4.2. Struktura attribut_t

Tato struktura reprezentuje záznam jednoho XML atributu.

```
struct attribut_t
{
    char * aname  ;
    char * avalue ;
    attribut_t * next ;
    attribut_t(const char * aname, const char * avalue);
    ~attribut_t();
    attribut_t * add_member(const char * aname, const char * avalue);
    void print(char * msg);
};
```

Struktura je záměrně implementována jako spojový seznam atributů. Toto supluje funkci jakou zastává pole, ale s menšími paměťovými nároky.



Příklad 4–2 Spojový seznam 4 atributů

4.2.1. Popis členů a metod

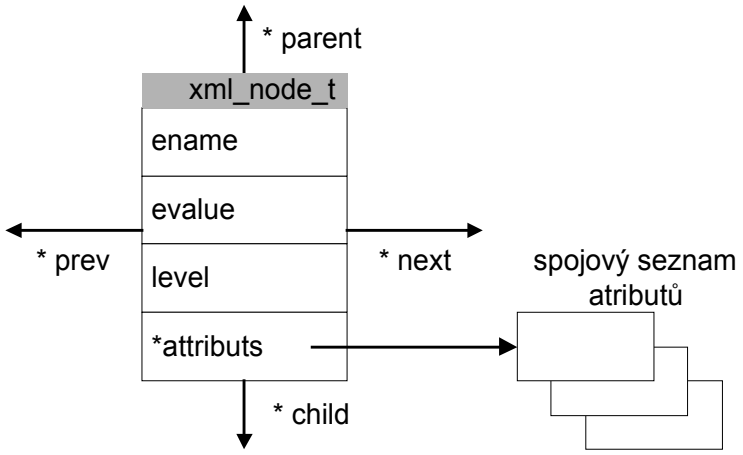
aname	jméno atributu
avalue	hodnota atributu
* next	další položka spojového seznamu
attribut_t(aname, avalue)	konstruktor vytvoří atribut daného jména s danou hodnotou
add_member(aname, avalue);	připojí nový atribut na konec seznamu
print(msg)	výpis traceinfo

Tabulka 4–1 attribut_t: členy a metody

4.3. Struktura xml_node_t

Tato struktura reprezentuje záznam jednoho XML uzlu. Uzel je chápán jako jeden element stromové struktury XML souboru.

```
struct xml_node_t
{
    char * ename ;
    char * evalue ;
    int level ;
    attribut_t * attributs;
    xml_node_t * prev ;
    xml_node_t * next ;
    xml_node_t * child ;
    xml_node_t * parent ;
    xml_node_t(const char * ename,
               const char * evalue,
               int level,
               xml_node_t * prev,
               xml_node_t * parent);
    ~xml_node_t();
    void set_value(const char * evalue);
    xml_node_t  * add_member( const char * ename, const char * evalue) ;
    xml_node_t  * add_child(  const char * ename, const char * evalue) ;
    attribut_t * add_attribut(const char * aname, const char * avalue) ;
    void print(char * msg);
};
```



Obrázek 4.1 Struktura xml_node_t

4.3.1. Popis členů a metod

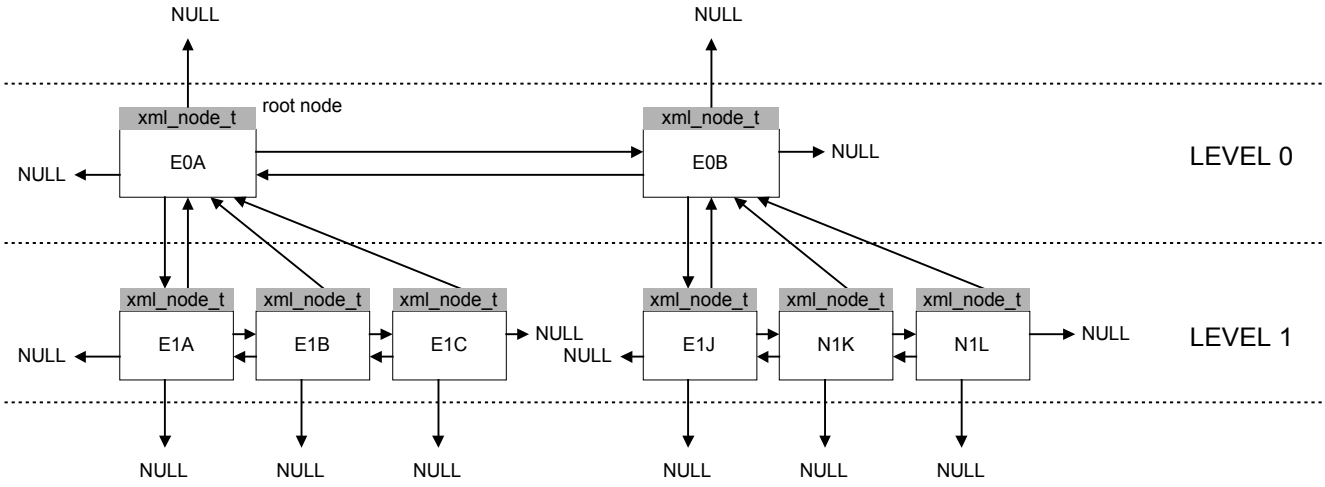
ename	jméno elementu
evalue	hodnota elementu
level	úroveň ve stromové struktuře
* attributs	spojový seznam atributů
* prev	uzel vlevo
* next	uzel vpravo
* child	uzel pod
* parent	uzel nad
xml_node_t(ename, evalue, level, prev, parent)	konstruktor
set_value(evalue)	nastaví hodnotu elementu

add_member(ename, evalue)	přidá uzel do stejného levelu
add_child(ename, evalue)	přidá uzel do levelu+1
add_attribut(aname, avalue)	přidá atribut do spojového seznamu atributů
print(msg)	výpis traceinfo

Tabulka 4–2 xml_node_t: položky a metody

Následuje jednoduchý příklad XML souboru a XML stromu, který mu odpovídá:

```
<E0A>
  <E1A>value</E1A>
  <E1B>value</E1B>
  <E1C>value</E1C>
</E0A>
<E0B>
  <E1J>value</E1J>
  <E1K>value</E1K>
  <E1L>value</E1L>
</E0B>
```



Obrázek 4.2 XML strom

Metody add_member a add_child přidávají uzly vždy **na konec seznamu**. Příklad:

E0A.add_member(...) přidá uzel na pozici E0C
E0A.add_child(...) přidá uzel na pozici E1D
E0B.add_child(...) přidá uzel na pozici E1M
E1A.add_child(...) přidá uzel na pozici E2A

4.4. Funkce build_xml_tree

deklarace: `xml_node_t * build_xml_tree(FILE * xmlfid)`

Procedura obdrží ukazatel na XML soubor, podrobí ho rozkladu, vytvoří XML strom a vrátí ukazatel na root uzel. Využívá vlastní implementaci funkcí StartElementHandler, DefaultHandler, EndElementHandler. Následuje příklad zpracování jednoduchého XML souboru:

XML file	XML parser
<element_0A>	StartElementHandler("element_0A")
<element_1A>	StartElementHandler("element_1A")
<element_2A>	StartElementHandler("element_2A")
value_2A	DefaultHandler("value_2A")
</element_2A>	EndElementHandler("element_2A")
</element_1A>	EndElementHandler("element_1A")
<element_1B>	StartElementHandler("element_1B")
value_1B	DefaultHandler("value_1B")
</element_1B>	EndElementHandler("element_1B")
</element_0A>	EndElementHandler("element_0A")

Příklad 4–3 Zpracování jednoduchého XML souboru

Z uvedeného příkladu lze vyvodit několik závěrů:

- nový XML uzel je třeba vytvořit při každém vstupu do funkce StartElementHandler
- pokud vstupujeme do funkce StartElementHandler dvakrát po sobě, posuneme se do dalšího levelu (level → level+1)
- pokud vstupujeme do funkce EndElementHandler dvakrát po sobě, vracíme se do předchozího levelu (level → level-1)

Algoritmus rozkladu XML souboru bude tedy fungovat následovně:

- zavedeme proměnnou **cnode**, která bude odkazovat na právě aktivní XML uzel
- zavedeme proměnnou **start_element flag**, kterou budeme odchytávat přechod do dalšího levelu
- zavedeme proměnnou **end_element flag**, kterou budeme odchytávat návrat do předchozího levelu

Proměnná cnode se může s výhodou předávat XML funkcím přes argument userData, viz. kapitola 4.1. Proměnné start_element_flag a end_element_flag se budou předávat XML funkcím jako globální proměnné. Kostra algoritmu build_xml_tree je tedy následující :

```
xml_node_t * cnode = NULL ;
XML_SetStartElementHandler( xml_parser, StartElementHandler);
XML_SetDefaultHandler(      xml_parser, DefaultHandler);
XML_SetEndElementHandler(   xml_parser, EndElementHandler);
XML_SetUserData(            xml_parser, (void *) &cnode);

start_element_flag = false ;
end_element_flag   = false ;

do
{
  char buff[BUFF_SIZE];
  int len = fread(buff, 1, sizeof(buff), xmlfid);
  XML_Parse(xml_parser, buff, len, feof(xmlfid)) ;
}
while( !feof(xmlfid) );
```

Příklad 4–4 Funkce build_xml_tree

Zde je uveden stručný popis implementace XML funkcí:

```
if( cnode == NULL )
{
  /* ----------------------------- */
  /* ----- make the root node ---- */
  /* ----------------------------- */

  cnode = new xml_node_t(ename, NULL, 0, NULL, NULL);
}
else if( start_element_flag )
{
  /* ------------------------- */
  /* ----- make new child ---- */
  /* ------------------------- */

  cnode = cnode->add_child(ename, NULL);
}
else
{
  /* -------------------------- */
  /* ----- make new member ---- */
  /* -------------------------- */

  cnode = cnode->add_member(ename, NULL);
}

// flags of level transition indication
start_element_flag = true  ;
end_element_flag   = false ;
```

Příklad 4–5 Funkce StartElementHandler

Funkce DefaultElementHandler je triviální:

```
cnode->set_value( s );
```

```
// leaving the level
if( end_element_flag )
  cnode = cnode->parent;  // move back to parent node

// flags for level transition indication
start_element_flag = false ;
end_element_flag   = true  ;
```

Příklad 4–6 Funkce EndElementHandler

Výše uvedené výpisy jsou velmi zjednodušené, neobsahují např. načítání atributů, ochranu proti vstupu do prázdného elementu, atp.

4.5. Procedura find_xml_nodes

deklarace :
```
void find_xml_nodes(xml_node_t *  inode,
                    xml_node_t ** rnodes,
                    int        *  rcount,
                    int    level,
                    char * ename,
                    char * evalue,
                    char * a0name  = NULL,
                    char * a0value = NULL,
                    ...
                    char * a7name  = NULL,
                    char * a7value = NULL);
```

Funkce prohledá XML strom od uzlu `*inode` a v proměnné `*rnodes[]` vrací výsledky jako pole ukazatelů na uzly, které odpovídají zadanému filtru. Počet nalezených uzlů je uložen v proměnné `count`. Lze volit následující filtry prohledávání:

a) hledání uzlu pouze v definovaném levelu (proměnná level)
b) pro level = –1 se prohledávají všechny levely
c) lze prohledávat jméno a hodnotu elementu
d) lze prohledávat hodnoty atributů a0 až a7
e) pokud se má některý z char* filtrů ignorovat, lze zadat konstantu NULL

Algoritmus prohledávání demonstruje následující pseudokód:

```
if( inode->level  == level &&
    inode->ename  == ename &&
    inode->evalue == evalue &&
    ...)
    // nalezena shoda
    rnodes[(*rcount)++] = inode ;

// prohledávání do hloubky
find_xml_nodes(inode->child)

// prohledávání do šířky
find_xml_nodes(inode->next)
```

4.6. Variabilní datový typ VARIANT

VARIANT umožňuje „měnit“ svůj datový typ za běhu programu. Může nabývat těchto hodnot:

- VT_UI1, VT_UI2, VT_UI4
- VT_I1, VT_I2, VT_I4
- VT_R4, VT_R8
- VT_BOOL
- další nás nezajímají

Definice VARIANTu je následující:

```
struct VARIANT
{
     VARTYPE vt;

     union
     {
       unsigned char   bVal;   // VT_UI1  VARTYPE=17
       unsigned short uiVal;   // VT_UI2  VARTYPE=18
       unsigned long  ulVal;   // VT_UI4  VARTYPE=19
       char            cVal;   // VT_I1   VARTYPE=16
       short           iVal;   // VT_I2   VARTYPE=2
       long            lVal;   // VT_I4   VARTYPE=3
       float         fltVal;   // VT_R4   VARTYPE=4
       double        dblVal;   // VT_R8   VARTYPE=5
       VARIANT_BOOL  boolVal;  // VT_BOOL VARTYPE=11
    ...další
    }
}
```

VARIANT se musí před prvním použitím inicializovat:

```
VARIANT      Substitute ;
VariantInit( &Substitute );
fill_variant("28.3", VT_R4, Substitute);
```

Protože ICBA metoda *ICBAAccoMgt::AddConnections()* požaduje ve dvou svých argumentech (konkrétně SubstituteValue a EpsilonValue) proměnnou typu VARIANT, byla napsána funkce **fill_variant**, která konvertuje ze stringu do VARIANTu:

```
int fill_variant(char * str, VARTYPE type, VARIANT * var)
```

Funkce podporuje pouze datové typy uvedené na začátku kapitoly. Pro konverzi do VT_BOOL musí být vstupní string malými písmeny, tedy „false" nebo „true". Pro interpretaci návratového kódu jsou v souboru `vartype.h` definována následující makra:

```
#define OK                 0
#define NOT_NUMERIC_VALUE  1  // string neobsahuje číselný údaj
#define NOT_BOOL_VALUE     2
#define OUT_OF_BOUNDARIES  3  // vstupní hodnota je mimo meze datového typu
#define UNKNOWN_DATATYPE   4  // nepodporovaný datový typ
```

4.7. Definice portu – struktura port_t

```
struct port_t
{
    char *          pname ;
    pdir_t          pdir  ;
    VARENUM         ptype ;
    cba_device_t * pdevice ;
    port_t(xml_node_t * pnode, cba_device_t * pdevice, bool cba_node);
    port_t(xml_node_t * pnode, cba_device_t * pdevice, int  proj_node);
    ~port_t();
    char * ptype2string();
    char * pdir2string();
    void print(char * msg);
};
```

4.7.1. Popis členů a metod

pname	jméno portu
pdir	orientace portu (1 = vstup, 2 = výstup.)
ptype	datový typ portu (viz. 2.7.6)
* pdevice	komponenta která daný port vlastní
port_t(* pnode, * pdevice, bool cba_node)	konstruktor z XML uzlu ve formátu CBA XML
port_t(* pnode, * pdevice, int proj_node)	konstruktor z XML uzlu ve formátu KIMAP XML
char * ptype2string()	převede ptype na string
char * pdir2string()	převede pdir na string
void print(msg)	výpis traceinfo

Tabulka 4–3 port_t: členy a metody

Metoda ptype2string() převádí datový typ portu na string, tj. vrátí např. "VT_UI1".
Metoda pdir2string() převádí orientaci portu na string, tj. vrací „IN" nebo "OUT".
Tyto dvě metody jsou využívány pro výpisu traceinfo.

4.7.2. „CBA XML“ konstruktor

Tento konstruktor obdrží v parametru *pnode ukazatel na sekci z CBA XML souboru s definicí portu viz. Obrázek 2.26. Z této sekce extrahuje položky pname, pdir, ptype následovně:

1. vyhledej tento uzel: `ename=“property“, attribut[0]=<“name“, “Name“>`
2. pname = evalue
3. vyhledej tento uzel: `ename=“property“, attribut[0]=<“name“, “Direction“>`
4. pdir = evalue
5. vyhledej tento uzel: `ename=“property“, attribut[0]=<“name“, “Datatype“>`
6. ptype = evalue

4.7.3. „KIMAP XML“ konstruktor

Tento konstruktor obdrží v parametru *pnode ukazatel na sekci z KIMAP XML souboru s definicí portu viz. 0. Z tohoto uzlu extrahuje položky pname, pdir, ptype následovně:

1. pname = attribut[0].avalue
2. ptype = attribut[1].avalue
3. ptype = attribut[2].avalue

4.8. Definice CBA komponenty – struktura cba_device_t

```
struct cba_device_t
{
    char * Device_name ;
    char * LDev_name   ;
    char * RTAuto_name ;
    BYTE PDev_ipaddr3 ;
    BYTE PDev_ipaddr2 ;
    BYTE PDev_ipaddr1 ;
    BYTE PDev_ipaddr0 ;
    port_t * ports[PORTS_DEEP] ;
    int      ports_count ;
    ICBAPhysicalDevice * pICBAPhysicalDevice;
    ICBALogicalDevice  * pICBALogicalDevice  ;
    ICBAAccoMgt        * pICBAAccoMgt        ;
    cba_device_t(xml_node_t * dnode, bool cba_xml );    // cba_xml  code
    cba_device_t(xml_node_t * dnode, int  proj_xml );   // proj_xml code
    ~cba_device_t();
    bool initCBA();
    void clearCBA();
    void print(char * msg);
};
```

4.8.1. Seznam členů a metod

PDev_ipaddr3, 2, 1, 0	IP adresa
ports[]	pole portů
ports_count	počet portů
pICBAPhysicalDevice	ukazatel na rozhraní *ICBAPhysicalDevice*
pICBALogicalDevice	ukazatel na rozhraní *ICBALogicalDevice*
pICBAAccoMgt	ukazatel na rozhraní *ICBAAccoMgt*
cba_device_t(* dnode, bool cba_xml)	konstruktor z XML uzlu ve formátu CBA XML

cba_device_t(* dnode, int proj_xml)	konstruktor z XML uzlu ve formátu KIMAP XML
bool initCBA()	inicializuje přístup k rozhraním ICBAPhysicalDevice, ICBALogicalDevice, ICBAAccoMgt
void clearCBA()	uvolňuje instance rozhraní z paměti
print(msg)	výpis traceinfo

Tabulka 4–4 cba_device_t: členy a metody

4.8.2. „CBA XML“ konstruktor

Tento konstruktor obdrží v argumentu *dnode ukazatel na root uzel CBA XML souboru.

a) Extrakce položky Device_name probíhá následovně:

1. v uzlu dnode vyhledej uzel ESMaster_node:

```
level = 2
ename = “coclass“
attribut[0] = <“progid“,“CBAXMLServer.CBAXML“>
```

2. v uzlu ESMaster_node vyhledej tento uzel:

```
level=4
ename=“property“
attribut[0]=<“name“, “ObjectName“>
```

3. `Device_name = evalue`

Struktura ESMaster uzlu viz. Obrázek 2.20.

b) Extrakce položky LDev_name probíhá následovně:

4. v uzlu dnode vyhledej uzel ESDevice_node:

```
level = 2
ename = “coclass“
attribut[0] = <“progid“,“CBAESDeviceServer.CBAESDevice“>
```

5. z uzlu ESDevice_node vyhledej tento uzel:

```
level=4
ename=“property“
attribut[0]=<“name“, “ObjectName“>
```

6. `LDev_name = evalue`

Struktura ESDevice uzlu viz. Obrázek 2.22.

c) Extrakce položky RTAuto_name probíhá následovně:

7. v uzlu dnode vyhledej uzel ESAuto_node:

```
level = 2
ename = "coclass"
attribut[0] = <"progid","CBAESAutoServer.CBAESAuto">
```

8. v uzlu ESAuto_node vyhledej tento uzel:

```
level=4
ename="property"
attribut[0]=<"name", "ObjectName">
```

9. `LDev_name = evalue`

Struktura ESAuto uzlu viz. Obrázek 2.23.

d) Extrakce portů probíhá následovně:

10. v uzlu dnode vyhledej všechny uzly port_nodes:

```
level = 2
ename = "coclass"
attribut[0] = <"progid","CBAConnectionFacetServer.CBAConnector">
```

11. Každý uzel z port_nodes obsahuje CBA XML definici portu (viz. Obrázek 2.26). Z každého tohoto uzlu se vytvoří port voláním konstruktoru port_t(port_nodes[i], *pdevice, CBA_NODE)

4.8.3. „KIMAP XML" konstruktor

Tento konstruktor obdrží v parametru *pnode ukazatel na sekci z KIMAP XML souboru s definicí CBA zařízení viz. kapitola 0. Z tohoto uzlu se položky PDev_ipaddr3–0, Device_name, LDev_name, RTAuto_name extrahují následovně:

1. `Device_name = attribut[0].avalue`
2. `PDev_ipaddr_string =  attribut[1].avalue`
3. z řetězce PDev_ipaddr_string extrahuj jednotlivé adresy PDev_ipaddr3–0
4. `LDev_name = attribut[2].avalue`
5. `RTAuto_name = attribut[3].avalue`

4.8.4. Metoda initCBA()

Tato metoda vytvoří v paměti instance rozhraní *ICBAPhysicalDevice*, *ICBALogicalDevice*, *ICBAAccoMgt*. Jako první se vytvoří instance rozhraní *ICBAPhysicalDevice*, a to voláním systémové funkce:

```
CoCreateInstanceEx(CLSID_CBAPhysicalDevice,
                   NULL,
                   CLSCTX_ALL,
                   &ServerInfo,
                   1,
                   MultiQi);
```

CLSID_CBAPhysicalDevice – guid třídy implementující rozhraní *ICBAPhysicalDevice*
ServerInfo – IP adresa komponenty + nastavení RPC přístupových práv

MultiQi – definuje guid rozhraní *ICBAPhysicalDevice (IID)*

Zbylá dvě rozhraní se vytvoří voláním těchto metod:

```
pICBAPhysicalDevice->get_LogicalDevice(bstrLDevName, &pICBALogicalDevice);
pICBALogicalDevice->get_ACCO( (IUnknown **) &pICBAAccoMgt);
```

Nemělo by se zapomínat na kontrolu návratového kódu HRESULT. Pokud daná metoda proběhla bez chyby, pak návratový kód má hodnotu S_OK.

Jednotlivé metody rozhraní jsou k dipozici pomocí šipkové konvence – např. :

```
pICBAAccoMgt->AddConnections()
```

4.8.5. Metoda clearCBA()

Při uvolňování rozhraní z paměti se **nesmí** použít operátoru delete. Pro uvolnění instance rozhraní z paměti se vždy používá metoda *IUnknown::Release()*. Rozhraní se z paměti musí uvolňovat v opačném pořadí, tj. pICBAAccoMgt, pICBALogicalDevice, pICBAPhysicalDevice:

```
if( pICBAAccoMgt != NULL )
{
  pICBAAccoMgt->Release() ;
  pICBAAccoMgt = NULL ;
}

// totéž pro pICBALogicalDevice, pICBAPhysicalDevice
```

4.9. Definice CBA spojení – struktura connection_t

```
struct connection_t
{
    port_t * output ;  // provider port
    port_t * input  ;  // consumer port
    WORD          QoSValue ;
    WORD          QoSType ;
    VARIANT       Substitute ;
    VARIANT       Epsilon ;
    char *        strSubstitute ;
    char *        strEpsilon ;
    DWORD         ConsumerID ;  // information for ICBAAccoMgt::ClearConnection
    bool          state ;       // true=active connection
    connection_t(port_t *      output,
                 port_t *      input,
                 WORD          QoSValue,
                 WORD          QoSType,
                 VARIANT       Substitute,
                 VARIANT       Epsilon,
                 const char *  strSubstitute,
                 const char *  strEpsilon);
    ~connection_t();
    void print(char * msg);
};
```

4.9.1. Seznam členů a metod

output	výstupní port
input	vstupní port
QoSValue, QoSType	požadované QoS
Substitute, Epsilon	substitute_value resp. epsilon_value
strSubstitute, strEpsilon	totéž v podobě řetězce
ConsumerID	jedinečný identifikátor spojení
state	stav (true = datové spojení je aktivní)
connection_t(...)	konstruktor
print(msg)	výpis traceinfo

Tabulka 4–5 connection_t: členy a metody

4.10. KIMAP XML – ukládání KiMap projektu

Všechna nastavení (importované komponenty, datová spojení) lze uložit do projektového souboru ve formátu KIMAP XML. KIMAP XML se skládá ze dvou sekcí – **devices** a **connections**. Následuje krátký příklad uloženého projektu:

```
<kimapxml version="1.0">

    <devices>

        <device Device_name="Dopravnik1"
                PDev_ipaddr="147.32.86.8"
                LDev_name="Dopravnik1_LDev"
                RTAuto_name="Dopravnik1_RTAuto">
                <ports>
                    <port name="pwm" type="17" direction="1"></port>
                    <port name="odeber_event" type="11" direction="1"></port>
                    <port name="command" type="17" direction="1"></port>
                    <port name="otacky" type="17" direction="2"></port>
                    <port name="ready" type="11" direction="2"></port>
                    <port name="P0123" type="19" direction="2"></port>
                    <port name="P4567" type="19" direction="2"></port>
                </ports>
        </device>

        ..... další zařízení

    </devices>

    <connections>

        <connection provider="Dopravnik1"
                    output="otacky"
                    consumer="PLC"
                    input="D1_otacky_real"
                    QoSValue="100"
                    QoSType="ms"
                    Substitute="50"
                    Epsilon="0">
        </connection>

        ..... další datová spojení

    </connections>

</kimapxml>
```

KIMAP XML se ukládá včetně šablony, ale toho bohužel nemůže být využito, protože EXPAT XML Parser nepodporuje kontrolu obsahu XML souboru podle definované šablony. Výpis KIMAP XML šablony viz. Příloha N.

4.11. Konfigurace a rušení datových spojení

Tyto tři funkce jsou definovány v souboru acco_mgt.h, cpp a slouží ke konfiguraci datových spojení na základě informací, které jsou obsaženy ve strukturách cba_device_t, connection_t.

a) konfigurace a aktivace datového spojení, jehož popis je uložen ve struktuře connection:

```
bool acco_mgt_add_connections(connection_t * connection)
```

b) zrušení datového spojení:

```
bool acco_mgt_remove_connections(connection_t * connection)
```

c) zrušení všech datových spojení daného zařízení:

```
bool acco_mgt_clear_connections(cba_device_t * device)
```

Funkce vrací true v případě korektního vykonání, jinak false.

4.11.1. ICBAAccoMgt::AddConnections

Tato metoda rozhraní slouží k vytvoření nového datového spojení. Podrobný popis viz. [1] kapitola 2.18.13.1. Volá se na straně consumera a má následující deklaraci:

```
HRESULT AddConnections(Provider, QoSType, QoSValue, State, Count,
                       ADDCONNECTIONIN[], ADDCONNECTIONOUT[]);
```

Provider – řetězec identifikující LDev zařízení providera ve tvaru "`PDev!LDev`"
– např. "`147.32.86.26!Dopravnik1_LDev`"
QoSType, QoSValue – požadovaná QoS
State – true = active, false = inactive spojení
Count – počet spojení definovaný v ADDCONNECTIONIN poli
ADDCONNECTIONIN[] – pole velikosti Count, každá položka definuje parametry spojení
ADDCONNECTIONOUT[] – pole velikosti Count, návratová proměnná

```
typedef struct ADDCONNECTIONIN
{
  ProviderItem;     // řetězec identifikující výstupní port providera ve tvaru "RTAuto.port"
                    // např. "Dopravnik1_RTAuto.otacky"
  ConsumerItem;     // řetězec identifikující vstupní port consumera ve tvaru "RTAuto.port"
                    // např. "Monitor_RTAuto.D1_otacky_real"
  PERSISTDEF;       // persistence (0 = CBAVolatile, 1 = CBAPendingPersistent, 2 = CBAPersistent)
  SubstituteValue;// substitute_value v podobě VARIANT proměnné
  Epsilon;          // epsilon_value v podobě VARIANT proměnné
};

typedef struct ADDCONNECTIONOUT
{
  ConsumerID;       // jedinečný identifikátor navázaného spojení; bude použito při rušení spojení
```

```
  Version;
  ErrorState;    // návratová hodnota typu HRESULT
};
```

4.11.2. ICBAAccoMgt::RemoveConnections

Tato metoda rozhraní slouží k rušení stávajících datových spojení. Podrobný popis viz. [1] kapitola 2.18.13.2. Volá se na straně consumera a má následující deklaraci:

```
HRESULT RemoveConnections(Count, ID[], Error[]);
```

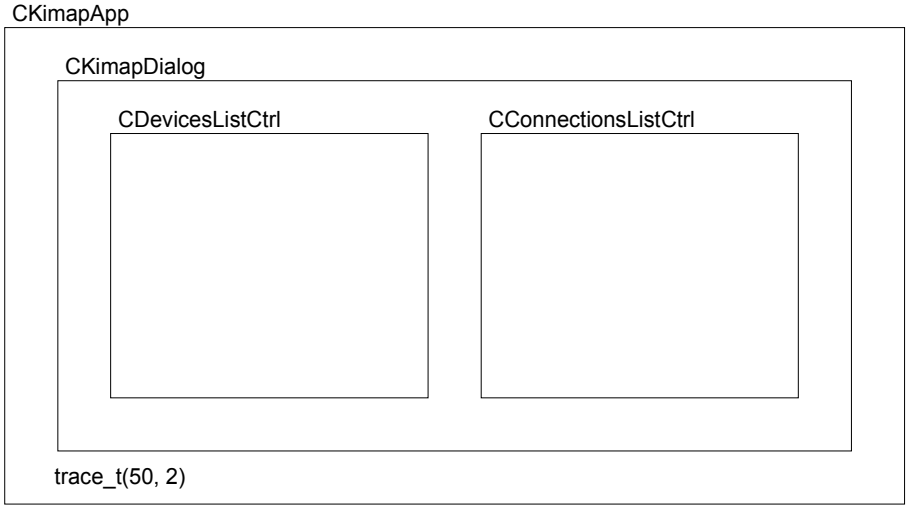
Count – počet spojení definovaných v ID argumentu
ID[] – pole velikosti Count, každá položka definuje identifikátor rušeného spojení, tj ConsumerID přijaté při navazování spojení
Error[] – pole velikosti Count, návratové hodnoty typu HRESULT

4.11.3. ICBAAccoMgt::ClearConnections

Tato metoda rozhraní slouží k rušení všech nakonfigurovaných spojení daného zařízení. Podrobný popis viz. [1] kapitola 2.18.13.3. Deklarace:

```
HRESULT ClearConnections();
```

4.12. Grafické prostředí



Obrázek 4.3 KiMap – objektová struktura grafického prostředí

Uveden bude pouze velmi stručný popis grafického rozhraní, protože není stěžejní částí aplikace. Reálný vzhled KiMapu viz. Příloha K, Příloha L, Příloha M.

Třída CKimapDialog implementuje všechny funkce obsluhující grafické prostředí – tj. funkce reagující na stisk tlačítek, zobrazení dialogových oken, kontrolu parametrů zadaných v dialozích, atd. CDevicesListCtrl a CConnectionsListCtrl jsou seznamy třídy CListCtrl. Pro uložení CBA zařízení nebo datového spojení do těchto seznamů se používá následující postup:

Při vkládání nového zařízení se do lokálních proměnných třídy CDevicesListCtrl vloží adresa instance tohoto zařízení:

```
cba_device_t * device = new cba_device_t(...);
CDevicesListCtrl.SetItemData(index, (DWORD) &device);
```

Adresu instance zařízení je možné získat metodou GetItemData:

```
cba_device_t * device = (cba_device_t *) CDevicesListCtrl.GetItemData(index);
```

Rušení zařízení a jeho uvolnění z paměti se provádí takto:

```
cba_device_t * device = (cba_device_t *) CDevicesListCtrl.GetItemData(index);
CDevicesListCtrl.DeleteItem(index);
delete device;
```

Pro vkládání resp rušení datového spojení se používá stejný postup.

Globální proměnná typu trace_t slouží k výpisu trasovacích informací.

4.13. Trasování a zpracování výjimek

V Kimapu je implementován vlastní trasovací mechanismus a zpracování výjimek. Pokud při spuštění KiMapu zadáme parametr –t, pak jsou generovány následující soubory:

1. kimap_trace.el – výpis volaných funkcí včetně jejich parametrů
2. kimap_trace.wr – výpis zpráv typu Warning a Error
3. kimap_trace.c – výpis komentářových zpráv
4. kimap_trace.exc – výpis výjimek
5. kimap_trace.all – výpis všech trasovacích informací

```
void function(int arg1, int arg2)
{
    char level_msg[STRING_LENGTH] ;
    char log_msg  [STRING_LENGTH] ;

    sprintf(level_msg, "function(%d, %d)“, arg1, arg2);

    try
    {
        trace->enter(level_msg);

        if( /* warning condition */ )
        {
          sprintf(log_msg, “warning message“);
          trace->warning(log_msg);
        }

        if( /* error condition */ )
        {
          sprintf(log_msg, “error message“);
          trace->error(log_msg);
          trace->leave();
          return;
        }

        if( /* ERROR_EXC exception condition */ )
            throw new exc_t(ERROR_EXC, NULL, "error-exception message") ;

        if( /* FAIL_EXC exception condition */ )
            throw new exc_t(FAIL_EXC, NULL, "fail-exception message") ;

        /* make comment message */
        sprintf(log_msg, “function has finished succesfully“);
        trace->comment(log_msg);
        trace->leave();
    }
    catch( exc_t * e )
    {
        e->add_level(level_msg);
        trace->leave();
        throw ;
    }
    catch( ... )
    {
        trace->leave();
        throw new exc_t(ERROR_EXC, level_msg, "unhandled exception");
    }
}
```

Příklad 4–7 Ukázková funkce generující traceinfo a výjimky

Po vstupu do funkce je třeba vygenerovat level_msg, tj. řetězec obsahující název funkce a seznam všech jejích argumentů. Poté předat tuto informaci trasovacímu systému:

```
trace->enter(level_msg);
```

Opuštění funkce se indikuje trasovacímu systému voláním

```
trace->leave();
```

Zpráva typu Warning – jedná se o upozornění na skutečnost, která by neměla nastat, ale je programově ošetřena (např. funkce obdrží v argumentech ukazatel na NULL)

Zpráva typu Error – jedná se o chybu, která je sice programově ošetřena, ale způsobí zablokování programu (např. nelze otevřít soubor)

Při generování výjimky je třeba specifikovat její severity:

- **severity ERROR_EXC** – použít v případě, kdy výjimka vzniká vlivem programátorské chyby např. když funkce obdrží argument nesmyslné hodnoty

- **severity FAIL_EXC** – použít v případě výjimek házených operačním systémem (nepřidělení požadované paměti, ...)

Komentáře jsou informace nechybového charakteru, které informují programátora o běhu programu (např. o úspěšném vzniku důležité třídy, vstup do časově náročné části programu, úspěšném ukončení funkce, ...).

Ukázky výpisů trasovacích informací jsou přiloženy, viz. Příloha O až Příloha R.

5. Obsah CD

/bak	/expat	EXPAT XML Parser (exe instalátor zdrojových souborů)
	/rs	RS software (zip archívy stažené z www.profibus.com)
/doc	/automa	článek o profinetu z časopisu Automa
	/CP343–1PN	dokumentace k profinet kartě
	/DCOM	jednoduché články o základech DCOM
	/diplomka	dokumentace diplomové práce
	/expat	dokumentace k EXPAT XML Parser
	/iMap	dokumentace k Siemens iMap
	/rs	dokumentace k profinet RS
	/S7	dokumentace Simatic Manageru
	/S7–300	dokumentace PLC řady S7–300
	/spec	specifikace profinetu
	/xml	učební texty základů XML
/exe		zkompilované exe soubory VisualC++
/iMap		iMap knihovny a projekty
	/diplomka.cbl, cbp	projekt experimentálního modelu (dopravníky+robot+PLC+monitor)
	/template.cbl, cbp	projekt vzorového příkladu template (PST+PLCT)
/lib		zkompilované knihovny VisualC++ runtime knihovna CBARuntimePS.dll
/obj		zkompilované obj soubory VisualC++
/S7		Simatic Manager projekty
	/plc	projekt PLC z experimentálního modelu
	/plct	projekt PLCT (šablona projektu implementujícího profinet rozhraní)
/src		zdrojové kódy + VisualC++ workspace a projekty
	/cbatest	demonstrační aplikace implementující všechna CBA rozhraní
	/dopravnik1	zdrojové soubory + IDL popis
	/dopravnik2	zdrojové soubory + IDL popis
	/expat	zdrojové soubory EXPAT XML Parseru
	/kimap	zdrojové soubory
	/monitor	zdrojové soubory + IDL popis
	/robot	zdrojové soubory + IDL popis
	/pst	zdrojové soubory + IDL popis
	/rs	profinet RS
/tools		typelib–converter AMCvtTlb.exe registrační skripty pro aktivaci/deaktivaci MS DCOM
/xml	/diplomka	xml popisy všech komponent experimentálního modelu + KiMap projekt
	/template	xml popisy PST, PLCT + KiMap projekt

5.1. VisualC++ projekty

VisualC++ workspace (*.dsw) jsou umístěny v adresáři `/src`
VisualC++ projekty (*.dsp) jsou umístěny v adresáři `/src/jméno_projektu`
Zdrojové soubory daného projektu jsou umístěny v adresáři `/src/jméno_projektu`
Zkompilované obj–soubory jsou umístěny v adresáři `/obj/jméno_projektu`
Knihovny *.lib se linkují do adresáře `/lib`
Výsledné exe soubory se linkují do adresáře `/exe`
XML popisy komponent jsou umístěny v adresáři /xml

Důvodem pro toto členění je snaha o důsledné oddělení zdrojových a zkompilovaných souborů, což bylo výhodné ze zálohovacích důvodů v průběhu práce na diplomové práci (zálohují se zpravidla jen zdrojové soubory).

Ve VisualC++ lze vytvářet projekty (*.dsp) a tyto projekty sdružovat do workspace (*.dsw). Důvodem tohoto způsobu organizace je umožnit definování závislostí mezi jednotlivými projekty. Obsah workspace je ve VisualC++ zobrazen v okně Workspace window (View → Workspace). Pro kompilaci a slinkování daného projektu je nutné ho označit jako aktivní (kliknout na tento projekt ve Workspace window a přes pravé tlačítko myši vybrat Set as Active Project). Pak spustit příkaz Build (F7).

5.1.1. Workspace libraries.dsw

Jednotlivé projekty představují profinet runtime knihovny:

1. Acco.dsp
2. automarshal.dsp
3. DCom.dsp
4. Device.dsp
5. rpc.dsp
6. Trace.dsp

5.1.2. Workspace expat.dsw

Tento workspace obsahuje různé formy knihoven „Expat XML Parser“ balíku + demonstrační příklady:

1. elements.dsp (demonstrační příklad)
2. expat.dsp (dynamická knihovna do DLL)
3. expat_static (statická knihovna LIB)
4. expatw.dsp (dynamická knihovna DLL)
5. expatw_static.dsp (statická knihovna LIB)
6. outline.dsp (demonstrační příklad)
7. xmlwf.dsp (demonstrační příklad)

Projekty 2. , 3. používají pro reprezentaci znaku klasický datový typ char, projekty 4. 5. používají wchar_t. Použití typu wchar_t vede zřejmě ke generování rychlejšího kódu.

5.1.3. Workspace diplomka.dsw

Tento wokspace obsahuje profinet–server komponenty experimentálního modelu tj. projekty:

1. Dopravnik1.dsp
2. Dopravnik2.dsp
3. Monitor.dsp
4. Robot.dsp

Aplikace se linkují s profinet runtime knihovnami, proto je třeba nejdříve zkompilovat všechny projekty z kapitoly 5.1.1.

5.1.4. Workspace kimap.dsw

Obsahuje profinet–klient aplikaci KiMap. Aplikace se linkuje s knihovnou expat.lib, proto je třeba nejdříve zkompilovat projekt expat_static – viz. kapitola 5.1.2.

5.1.5. Workspace pst.dsw

Obsahuje vzorovou aplikaci typu profinet–server. Jméno projektu je odvozeno od zkratky PST (profinet–server template). Před vytvořením vlastní aplikace je třeba modifikovat soubory main.c, sysobj.c, idl.idl a PST.xml, jak je popsáno v kapitolách 2.9.3, 2.11.3, 2.12.5. Aplikace se linkuje s profinet runtime knihovnami, proto je třeba nejdříve zkompilovat všechny projekty z kapitoly 5.1.1.

5.1.6. Workspace CBAtest.dsw

CBATest je aplikace typu profinet–klient, která implementuje všechna CBA rozhraní. Umožňuje v grafickém prostředí přistupovat k CBA komponentám, vyčítat diagnostické informace, konfigurovat datová spojení atd. Jedná se o výborný prostředek pro seznámení se s funkcemi profinetu. Je součástí profinet RS.

5.2. Simatic Manager projekty

5.2.1. PLC

Tento projekt obsahuje komponentu PLC z experimentálního modelu.

5.2.2. PLCT

Šablona projektu implementujícího profinet komponentu na PLC CPU315–2DP+CP343–1PN. Jméno projektu je odvozeno od zkratky PLCT (PLC template). Tato šablona má přednastavené bloky FC56, FC57, OB1 a DB100, které je nutno patřičně modifikovat pro vytvoření vlastní komponenty – viz. kapitoly 3.4.2, 3.4.3, 3.4.12, 3.4.13.

6. Závěr

V předkládané práci byl realizován jednoduchý model průmyslové linky. Komponenty Dopravník1,2, Robot a Monitor byly naprogramovány jako profinet–server aplikace využívající profinet Runtime Source. Tento softwarový balík se jeví subjektivně jako příliš komplikovaný, zejména z pohledu jeho čitelnosti – nepřehledná adresářová struktura, množství externích odkazů a maker. Při návrhu profinet–server aplikace je nutné popisovat její strukturu celkem na třech místech – v IDL, XML a sysobj.c souborech, což je velmi pracné vzhledem ke skutečnosti, že obsahují tytéž údaje. Všechny čtyři aplikace (Dopravník1,2, Robot i Monitor) pracují korektně a po svém spuštění poskytují svému okolí definované properties.

Jako pátá komponenta v modelu vystupuje PLC CPU315-2DP. Rozšířením o kartu CP343-1PN lze realizovat profinet rozhraní s minimálním úsilím. Z hlediska softwarového řešení stačí vložit do S7 projektu jeden funkční (FB88) a jeden datový blok. Z PLC se stane profinet komponenta až po downloadu kódu v iMapu. Pak lze přistupovat k CPU jako k plně funkční profinet komponentě.

Stěžejním nástrojem pro konfiguraci sítě je konfigurační nástroj, tedy program iMap firmy Siemens. Ten při pokusu o download datových spojení vypíše chybovou hlášku a skončí. Tento fakt byl reportován firmě Siemens se žádostí o pomoc, ale bohužel bez jakékoli odezvy. Protože download datových spojení je z hlediska ověření činnosti experimentálního modelu klíčová, byla naprogramována aplikace KiMap, která supluje funkci iMapu. KiMap načítá XML popis komponent, konfiguruje datová spojení a provádí jejich download. Jedná se o aplikaci typu profinet–klient. Nevyřešeným problémem zůstává DCOM–timeout – pokud v Kimapu vytvoříme DCOM instanci vzdálené komponenty a tato komponenta zanikne, pak se v případě pokusu o přístup k ní generuje timeout přibližně 2 minuty. Nebylo nalezeno řešení, jak dobu DCOM–timeout omezit. Jednou z nevyzkoušených možností je prostudovat DCOM záznam ve Windows registru a pokusit se najít položku, která tento timeout nastavuje.

Po provedení downloadu datových spojení prostřednictvím aplikace KiMap probíhala mezi jednotlivými komponentami datová výměna podle definovaných parametrů. Nejkratší perioda datové výměny, při které nedochází k výpadkům přenosu hodnot, je 100ms. Bylo ověřeno, že data se přenášejí pouze v případě, kdy jejich změna přesáhne definovanou hodnotu epsilon_value. Byl rovněž učiněn experiment, kdy jedna komponenta byla odpojena od sítě – ostatní pak namísto přerušených datových spojení dosazovaly hodnotu substitute_value. Lze tedy konstatovat, že v praktickém provozu datová výměna probíhá korektně a to včetně reakce na vznik kritických událostí.

V diplomové práci byly naprogramovány a otestovány profinet–servery (Dopravník1, 2, Robot, Monitor), profinet–klient (KiMap) a profesionální zařízení (CPU315-2DP+CP343-1PN). Všechny tři druhy komponent pracovaly bez problémů a lze tedy říci, že síť profinet, přestože se jedná o poměrně nový standard, nevykazuje žádné zjevné nedostatky.

Výsledky diplomové práce mohou být použity jako základ další práce s profinetem. Zejména pak aplikace KiMap, kde je poměrně široké pole působnosti – lze ho rozšířit o diagnostické funkce, online sledování přenášených dat, reakci na vznik chybových stavů komponent prostřednicvím asynchronních událostí, podporu relace LDev:RTAuto 1:n a download kódu. Lze se také pokusit najít řešení DCOM–timeout problému či rozšířit množinu podporovaných datových typů.

7. Literatura

[1] PNO: Profinet Architecture Description and Specification V0.91, dokumentační pdf na http://www.profibus.com/

[2] PNO: Profinet Implementation Guide V1.0, dokumentační pdf na http://www.profibus.com/

[3] Siemens: Component Based Automation – Configuring Plants with iMap, 07/2001, instalační disk iMap trial v1.1

[4] Siemens: Description of the CP 343-1 PN, dokumentační pdf z propagačního CD „SIMATIC NET Manual Collection“

[5] Microsoft: MSDN Microsoft Developer Network – dokumentační databáze (součást VisualC++)

[6] www.microsoft.com: DCOM Technical overview

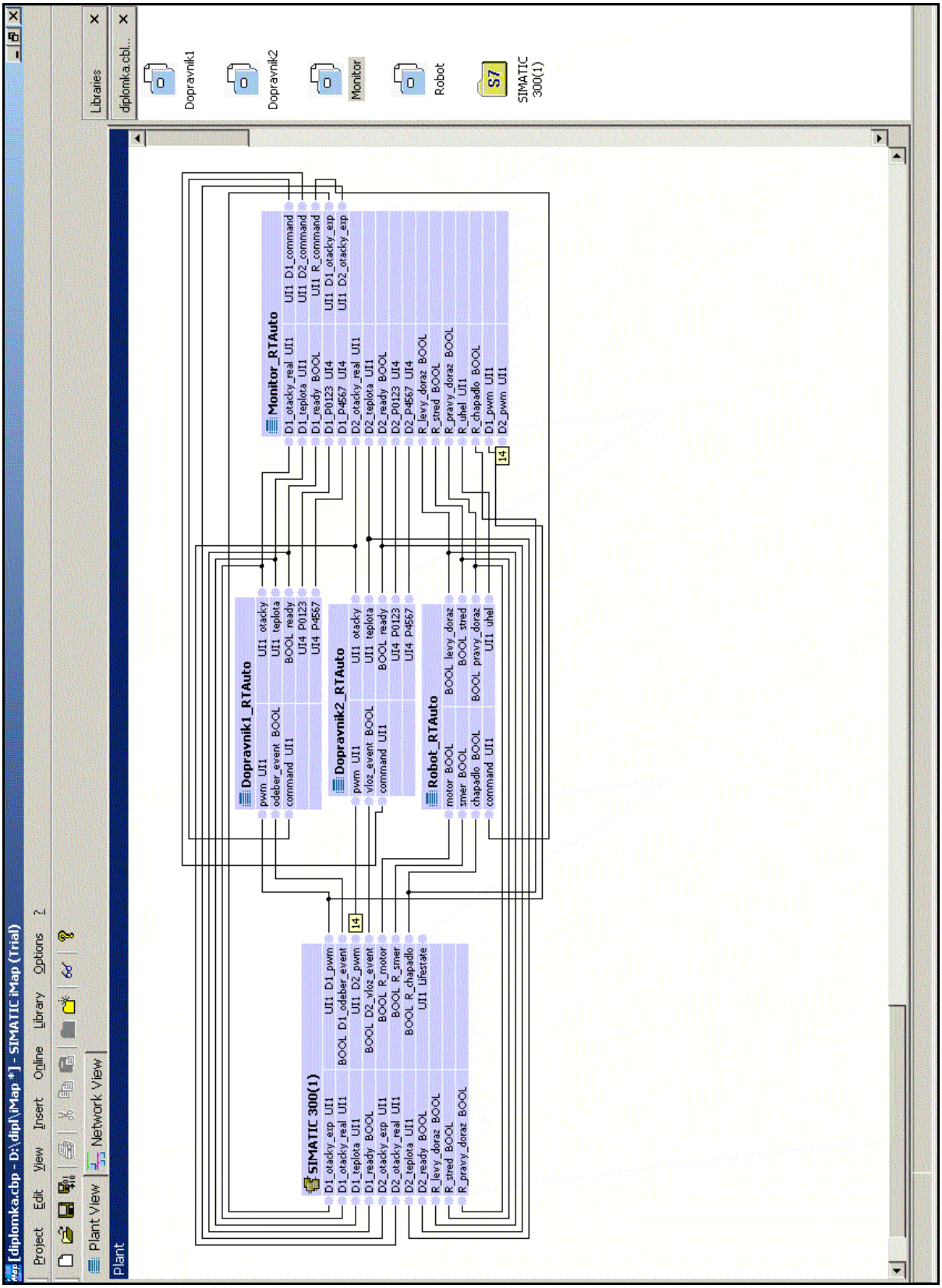
[7] www.microsoft.com: DCOM Architecture

[8] Grada: Ilja Kraval, Pavel Ivachiv: Základy komponentní technologie COM, Grada 1999

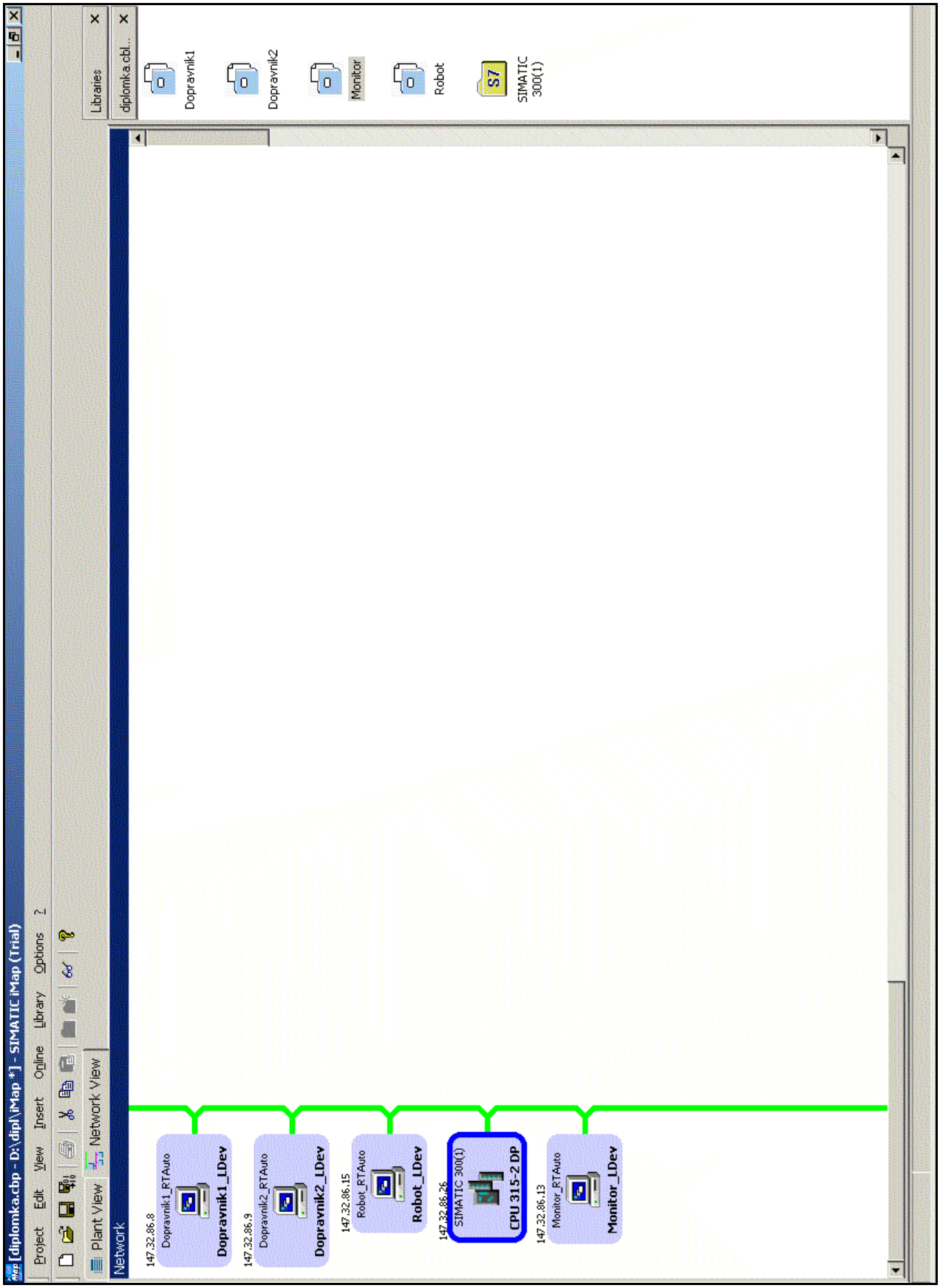
8. Přílohy

```
static const BYTE IDLLib_Tbl[]=
{
/*Dispatch Interface IPST*/
/*0000*/ 0x06,0x00, /*Number of Functions*/
/*0002*/ 0x0e,0x00, /*Offset of inputA 14*/
/*0004*/ 0x14,0x00, /*Offset of inputA 20*/
/*0006*/ 0x1e,0x00, /*Offset of inputB 30*/
/*0008*/ 0x24,0x00, /*Offset of inputB 36*/
/*0010*/ 0x2e,0x00, /*Offset of outputC 46*/
/*0012*/ 0x38,0x00, /*Offset of outputD 56*/
/*function inputA*/
/*0014*/ 0x01,PROPPUT|HAS_NAME, /*Number of Args, Flags*/
/*0016*/ 0x07,0x00,
/*0018*/ DCOM_VT_UI1,MODE_IN, /* pVal */
/*function inputA*/
/*0020*/ 0x01,PROPGET|HAS_NAME, /*Number of Args, Flags*/
/*0022*/ 0x0a,0x00,
/*0024*/ DCOM_VT_PTR,MODE_OUT|HAS_LENGTH, /*Unique Pointer pVal */
/*0026*/ 0x06,0x00,     /*Length of Elem in Bytes */
/*0028*/ DCOM_VT_UI1,NO_FLAGS, /*  */
/*function inputB*/
/*0030*/ 0x01,PROPPUT|HAS_NAME, /*Number of Args, Flags*/
/*0032*/ 0x0f,0x00,
/*0034*/ DCOM_VT_UI1,MODE_IN, /* pVal */
/*function inputB*/
/*0036*/ 0x01,PROPGET|HAS_NAME, /*Number of Args, Flags*/
/*0038*/ 0x12,0x00,
/*0040*/ DCOM_VT_PTR,MODE_OUT|HAS_LENGTH, /*Unique Pointer pVal */
/*0042*/ 0x06,0x00,     /*Length of Elem in Bytes */
```

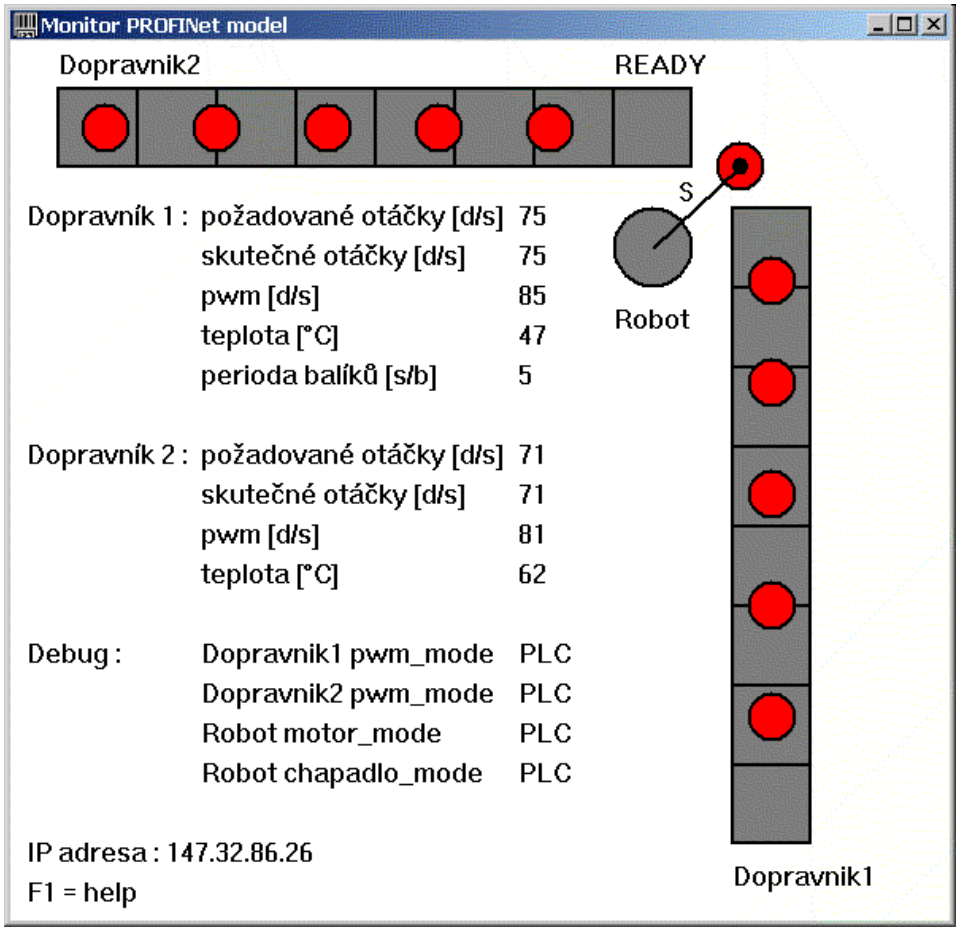
Příloha A Ukázka obsahu třídy IDLLib_Tbl



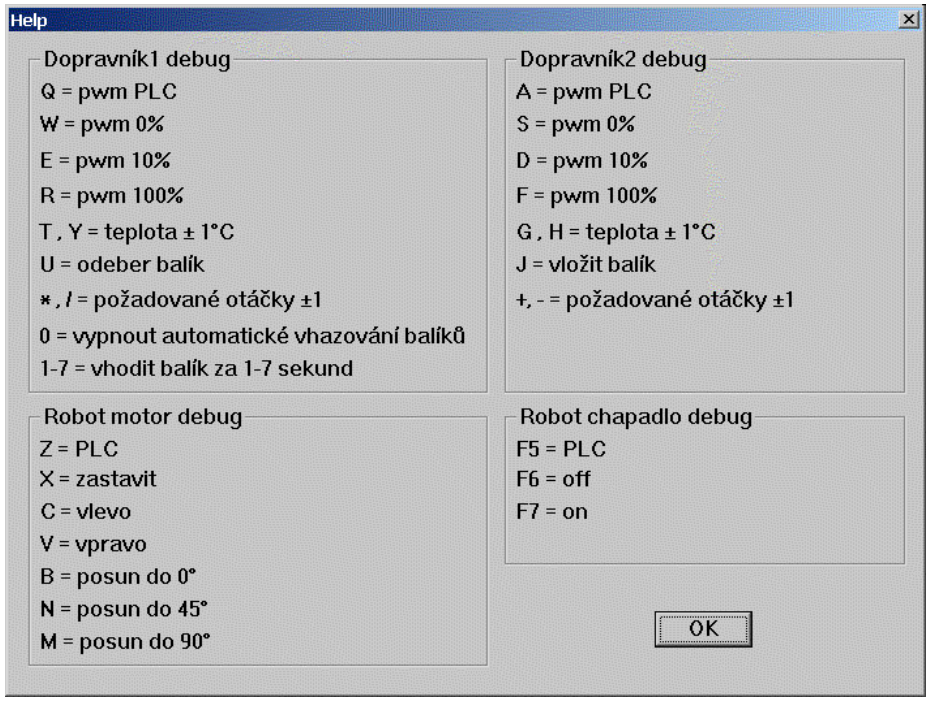
Příloha B iMap – Plant view



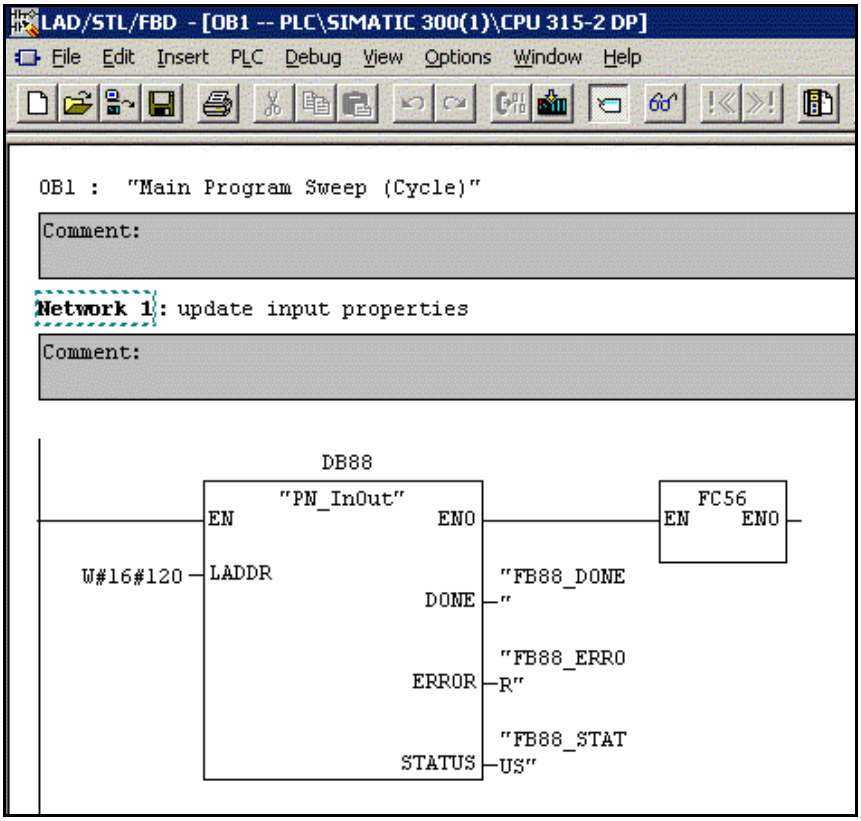
Příloha C iMap – Network view



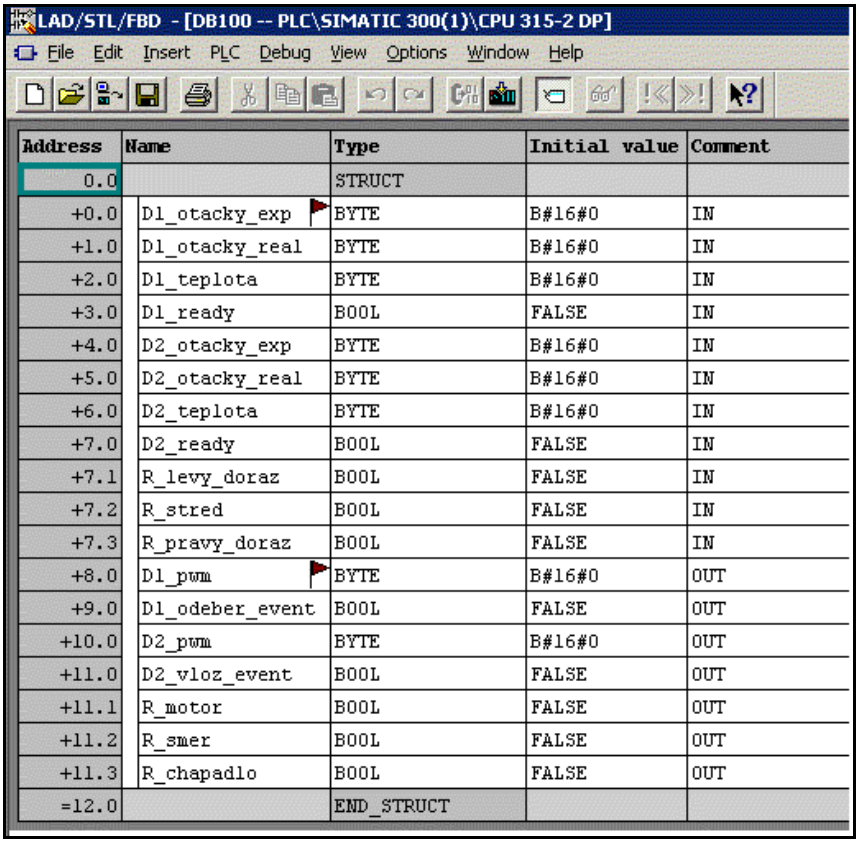
Příloha D Monitor – vizuální výstup



Příloha E Monitor – seznam klávesových zkratek



Příloha F PLC – příklad použití funkčního bloku FB88



Address	Name	Type	Initial value	Comment
0.0		STRUCT		
+0.0	D1_otacky_exp	BYTE	B#16#0	IN
+1.0	D1_otacky_real	BYTE	B#16#0	IN
+2.0	D1_teplota	BYTE	B#16#0	IN
+3.0	D1_ready	BOOL	FALSE	IN
+4.0	D2_otacky_exp	BYTE	B#16#0	IN
+5.0	D2_otacky_real	BYTE	B#16#0	IN
+6.0	D2_teplota	BYTE	B#16#0	IN
+7.0	D2_ready	BOOL	FALSE	IN
+7.1	R_levy_doraz	BOOL	FALSE	IN
+7.2	R_stred	BOOL	FALSE	IN
+7.3	R_pravy_doraz	BOOL	FALSE	IN
+8.0	D1_pwm	BYTE	B#16#0	OUT
+9.0	D1_odeber_event	BOOL	FALSE	OUT
+10.0	D2_pwm	BYTE	B#16#0	OUT
+11.0	D2_vloz_event	BOOL	FALSE	OUT
+11.1	R_motor	BOOL	FALSE	OUT
+11.2	R_smer	BOOL	FALSE	OUT
+11.3	R_chapadlo	BOOL	FALSE	OUT
=12.0		END_STRUCT		

Příloha G PLC – příklad datového bloku DB100

```
Network 1: Title:
Comment:

                                        "PN_Interf
                                        ace_DB"
                                          (OPN)

Network 2: "update D1_otacky_exp"
Comment:

      L     "PN_Interface_DB".D1_otacky_exp
      T     "D1_otacky_exp"

Network 5: "update D1_ready"
Comment:

 "PN_Interf
 ace_DB".D1
 _ready                                 "D1_ready"
 ---| |-------------------------------------( )---
```

Příloha H PLC – příklad funkce „update input properties“

```
Network 2: "update D1_odeber_event"
Comment:

                                        "PN_Interf
                                        ace_DB".D1
 "D1_odeber                             _odeber_ev
 _event"                                ent
 ---| |-------------------------------------( )---

Network 3: "update D2_pwm"

Comment:

  L     "D2_pwm"
  T     "PN_Interface_DB".D2_pwm
```

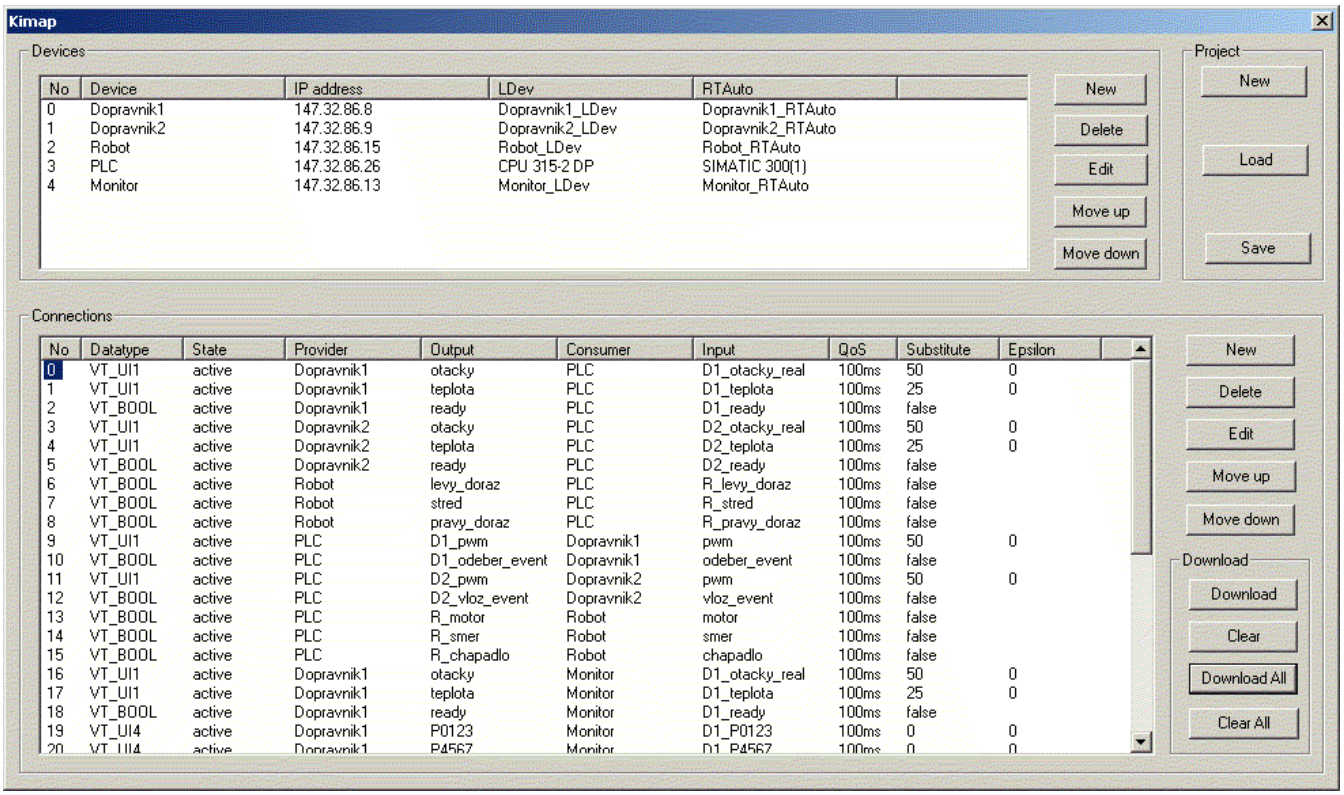
Příloha I PLC – příklad funkce „update output properties“

Monitoring and Modifying Variables - [@VAT -- diplomka\SIMATIC 300(1)\CPU 315-2 DP\S7 Program(1) ONLINE]

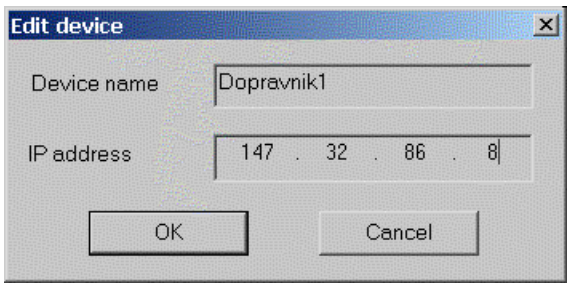
Table Edit Insert PLC Variable View Options Window Help

	Address	Symbol	Displ	Status value	Modify value
1	MB 0	"D1_otacky_exp"	DEC	30	
2	MB 4	"D1_otacky_real"	DEC	31	
3	MB 8	"D1_teplota"	DEC	25	
4	M 12.0	"D1_ready"	BIN	2#0	
5					
6	MB 16	"D1_pwm"	DEC	37	
7	M 20.0	"D1_odeber_event"	BIN	2#0	
8					
9	M 24.0	"D1_run"	BOOL	true	
10					
11	MB 32	"D2_otacky_exp"	DEC	100	
12	MB 36	"D2_otacky_real"	DEC	98	
13	MB 40	"D2_teplota"	DEC	25	
14	M 44.0	"D2_ready"	BOOL	true	
15					
16	MB 48	"D2_pwm"	DEC	100	
17	M 52.0	"D2_vloz_event"	BOOL	true	
18					
19	M 56.0	"D2_run"	BOOL	true	
20					
21	M 64.0	"R_levy_doraz"	BOOL	false	
22	M 68.0	"R_stred"	BOOL	false	
23	M 72.0	"R_pravy_doraz"	BOOL	false	
24					
25	M 76.0	"R_motor"	BOOL	true	
26	M 80.0	"R_smer"	BOOL	true	
27	M 84.0	"R_chapadlo"	BOOL	true	
28					
29	MB 88	"R_stav"	DEC	2	
30					
31	M 128.0	"FB88_DONE"	BOOL	true	
32	M 132.0	"FB88_ERROR"	BOOL	false	
33	MW 136	"FB88_STATUS"	HEX	W#16#0000	
34					

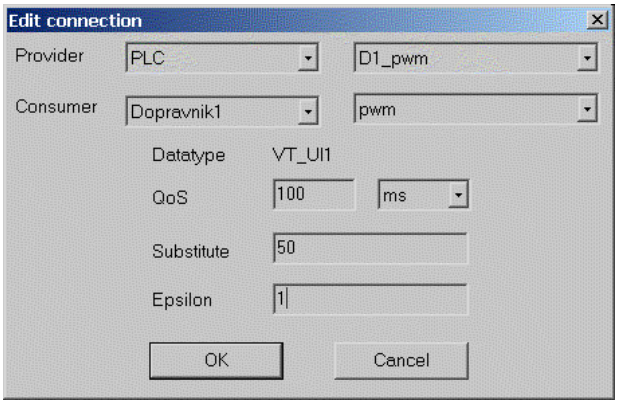
Příloha J PLC – mapování interních proměnných



Příloha K KiMap – hlavní dialogové okno



Příloha L KiMap – parametry komponenty



Příloha M KiMap – parametry datového spojení

```
<?xml version="1.0"?>
<!DOCTYPE kimapxml [

<!--  DTD of kimap XML project file  -->

<!ELEMENT kimapxml (devices, connections)>
<!ATTLIST kimapxml version CDATA #FIXED "1.0">

<!ELEMENT devices  (device*)>
<!ATTLIST device   Device_name CDATA #REQUIRED
                   PDev_ipaddr CDATA #REQUIRED
                   LDev_name   CDATA #REQUIRED
                   RTAuto_name CDATA #REQUIRED>
<!ELEMENT device (ports)>
<!ELEMENT ports (port*)>

<!ELEMENT connections (connection*)>
<!ATTLIST connection provider   CDATA #REQUIRED
                     output     CDATA #REQUIRED
                     consumer   CDATA #REQUIRED
                     input      CDATA #REQUIRED
                     QoSValue   CDATA #REQUIRED
                     QoSType    CDATA #REQUIRED
                     Substitute CDATA #REQUIRED
                     Epsilon    CDATA #REQUIRED>

<!-- End of the DTD -->
```

Příloha N KiMap – výpis KiMap DTD šablony

```
L    DefaultHandler("port", 4, "", 20)
E    StartElementHandler("port", 4, "port", "name", "R_levy_doraz")
E     xml_node_t(00E59DB0)::add_member("port", "NULL")
E      xml_node_t(00E59B00)::xml_node_t("port", "NULL", 4, 00E59DB0, 00E57170)
L      xml_node_t(00E59B00)::xml_node_t("port", "NULL", 4, 00E59DB0, 00E57170)
L     xml_node_t(00E59DB0)::add_member("port", "NULL")
E     xml_node_t(00E59B00)::add_attribut("name", "R_levy_doraz")
E      attribut_t(00E59A80)::attribut_t("name", "R_levy_doraz")
L      attribut_t(00E59A80)::attribut_t("name", "R_levy_doraz")
L     xml_node_t(00E59B00)::add_attribut("name", "R_levy_doraz")
E     xml_node_t(00E59B00)::add_attribut("type", "11")
E      attribut_t(00E59A80)::add_member("type", "11")
E       attribut_t(00E599C0)::attribut_t("type", "11")
L       attribut_t(00E599C0)::attribut_t("type", "11")
L      attribut_t(00E59A80)::add_member("type", "11")
L     xml_node_t(00E59B00)::add_attribut("type", "11")
E     xml_node_t(00E59B00)::add_attribut("direction", "1")
E      attribut_t(00E59A80)::add_member("direction", "1")
E       attribut_t(00E59910)::attribut_t("direction", "1")
L       attribut_t(00E59910)::attribut_t("direction", "1")
L      attribut_t(00E59A80)::add_member("direction", "1")
L     xml_node_t(00E59B00)::add_attribut("direction", "1")
L    StartElementHandler("port", 4, "port", "name", "R_levy_doraz")
E    EndElementHandler("port", 4, "port")
```

Příloha O KiMap – výpis kimap_trace.el souboru

```
W
{
  CKimapApp::Run()
  CKimapDialog::OnLoadProject()
  build_xml_tree(005A72A8)
  StartElementHandler(root, "kimapxml", "version", "1.0")
  xml_node_t(00E528A0)::xml_node_t("kimapxml", "NULL", 0, 00000000, 00000000)

  evalue is NULL
}
W
{
  CKimapApp::Run()
  CKimapDialog::OnLoadProject()
  build_xml_tree(005A72A8)
  StartElementHandler("kimapxml", 0, "devices", "no attr", "1.0")
  xml_node_t(00E528A0)::add_child("devices", "NULL")
  xml_node_t(00E52660)::xml_node_t("devices", "NULL", 1, 00000000, 00E528A0)

  evalue is NULL
}
W
{
  CKimapApp::Run()
  CKimapDialog::OnLoadProject()
```

Příloha P KiMap – výpis souboru kimap_trace.wr

```
EEXC
{
  CKimapApp::Run()
  {
  }

  CKimapDialog::OnLoadProject()
  {
    unhandled exception
  }
}
```

Příloha Q KiMap – výpis souboru kimap_trace.exc

```
C
{
  cba_device_t 00FD75B8
  Device       "Monitor"
  PDev_ipaddr  147.32.86.13
  LDev         "Monitor_LDev"
  RTAuto       "Monitor_RTAuto"
  port         "D1_otacky_exp" OUT VT_UI1
  port         "D1_otacky_real" IN VT_UI1
  port         "D1_pwm" IN VT_UI1
  port         "D1_teplota" IN VT_UI1
  port         "D1_ready" IN VT_BOOL
  port         "D1_P0123" IN VT_UI4
  port         "D1_P4567" IN VT_UI4
  port         "D1_command" OUT VT_UI1
  port         "D2_otacky_exp" OUT VT_UI1
  port         "D2_otacky_real" IN VT_UI1
  port         "D2_pwm" IN VT_UI1
  port         "D2_teplota" IN VT_UI1
  port         "D2_ready" IN VT_BOOL
  port         "D2_P0123" IN VT_UI4
  port         "D2_P4567" IN VT_UI4
  port         "D2_command" OUT VT_UI1
  port         "R_motor" IN VT_BOOL
  port         "R_smer" IN VT_BOOL
  port         "R_chapadlo" IN VT_BOOL
  port         "R_command" OUT VT_UI1
  port         "R_levy_doraz" IN VT_BOOL
  port         "R_stred" IN VT_BOOL
  port         "R_pravy_doraz" IN VT_BOOL
  port         "R_uhel" IN VT_UI1
}
C
{
  find_xml_nodes(00E528A0, 0012E204, 0012E200, 2, "connection", "NULL",
"provider", "NULL", "output", "NULL", "consumer", "NULL", "input", "NULL",
"QoSValue", "NULL", "QoSType", "NULL", "Substitute", "NULL", "Epsilon", "NULL")
: 38 nodes have been found
}
C
{
  connection_t 00E52B30
  provider     "Dopravnik1!Dopravnik1_LDev"
  output       "Dopravnik1_RTAuto.otacky" VT_UI1 OUT
  consumer     "PLC!CPU 315-2 DP"
  input        "PLC.D1_otacky_real" VT_UI1 IN
  QoS          100ms
  Substitute   50
  Epsilon      0
}
```

Příloha R KiMap – výpis souboru kimap_trace.c