České vysoké učení technické v Praze

Fakulta elektrotechnická Katedra řídicí techniky

Diplomová práce

Průmyslová komunikace PROFInet

2004 Ondřej Netík

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 souvisejcích s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne: ..... Podpis: .....

Abstrakt

PROFInet je z pohledu návrhu technologie, konfigurovatelnosti a diagnostiky jednotlivých částí mocný nástroj pro řízení, k přenosu dat využívá standardu Ethernet a TCP-IP, nad nimi je implementována vrstva DCOM kterou je vytvořen objektový model PROFInetu. Linux je stabilní, časem prověřený a volně šiřitelný operační systém. Spojení PROFInetu a Linuxu dává dobrý základ pro použití v aplikacích pro řízení. Tato práce se věnuje adaptaci PROFInetu pod operační systém Linux a rozšiřuje ho o webové rozhraní pro přístup k objektům PROFInetu protokolem HTTP (webový prohlížeč) a o webové služby přístupné protokolem SOAP pro snadnější integraci do manažerských aplikací.

Abstract

PROFInet is powerfull tool for control from viewpoint of design a technologi, configuration and diagnostic. It used Etherned and TCP-IP standard, above them is implemented DCOM layer that create PROFInet model. Linux is stable, time verified and free distributed operating system. Association of PROFInet and Linux are good base for use in aplication of control. This work concered with implementation PROFInet server under Linux and extend it with web interface for access to object of PROFInet with HTTP protocol and webservices for access with SOAP protocol.

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Původní záměr této práce bylo adaptovat verzi 1.2 (popřípadě v2.0) PROFInet serveru do operačního systému Linux s hlavním zaměřením na implementaci *Softwarového Realtimu* (viz. [4]). Léto 2003 jsem strávil ve firmě IFAK v Magdeburgu, kde jsem měl na této implementaci pracovat. Z důvodu zdržení specifikace[1] PROFInet serveru v2.0 bylo od tohoto záměru upuštěno a po dohodě s vedoucím diplomové práce a konzultantem ve firmě IFAK se náplní celé práce stala implementace PROFInet serveru pod OS Linux, Webové rozhraní a Webové služby.

V úvodu celé práce bych chtěl čtenáře seznámit se základními vlastnostmi PROFInetu a postupem návrhu aplikace. Protože tyto skutečnosti byly dobře popsány v [1], je tato kapitola spíše určena k definici pojmů, upozornění na odlišnosti při použití a implementování PROFInet serveru a vlastní aplikace pod operačním systémem Linux.

V následující kapitole se zaměříme na popis adaptace PROFInet serveru z platformy operačního systému Windows do Linuxu. Postup koresponduje s obecným postupem popsaným v [2].

Další kapitola rozšiřuje stávající Linuxovou implementaci PROFInet serveru o web rozhraní, na kterou navazuje kapitola o webových službách, které slouží především k vizualizaci a ovlivnění základních parametrů serveru.

V poslední kapitole je ukázka aplikace na Linuxovém PROFInet serveru implementována na průmyslovém počítači, se značně omezenými prostředky - model akvária.

Touto prací bych chtěl rozšířit diplomovou práci [1], z které jsem čerpal základní informace. Dalšími informačními zdroji o adaptaci serveru byl především [2], obecné informace o PROFInet serveru jsou specifikovány v [3], specifikace webového rozhraní je popsána v [4].

V práci jsou zachovány některé anglické názvy, pro které neexistuje jednoznačný překlad do českého jazyka.

[1] kterou má na starosti organizace PROFIBUS International

Kapitola 1

PROFInet

1.1 Úvod

PROFInet klade větší důraz na návrh struktury a funkce celého řízeného procesu než na implementaci hardwarových částí. Komponenta (objekt) představuje funkční celek PROFInetu, jež s okolím komunikuje pomocí povinně implementovaných rozhraní dané specifikací [3].

Pro interakci mezi jednotlivými komponentami PROFInetu se využívá standardu DCOM[1] firmy Microsoft. Ten definuje nad každou komponentou jedno nebo více rozhraní, pomocí nichž se přistupuje k funkcím a datům. DCOM definuje způsob jak funkce vyvolávat, ale způsob jejich implementace je skryt.

1.2 Model PROFInetu

PROFInet Runtime model reprezentuje funkční celky, které jsou fyzicky přítomny v zařízení a přístupné z okolí pomocí definovaných rozhraní a metod.

PhysicalDevice (označeno PDev) reprezentuje hardware-počítač nebo PLC. Umožňuje přístup k síti pomocí IP adresy. Na každém zařízení existuje pouze jedno PhysicalDevice (obr. 1.1).

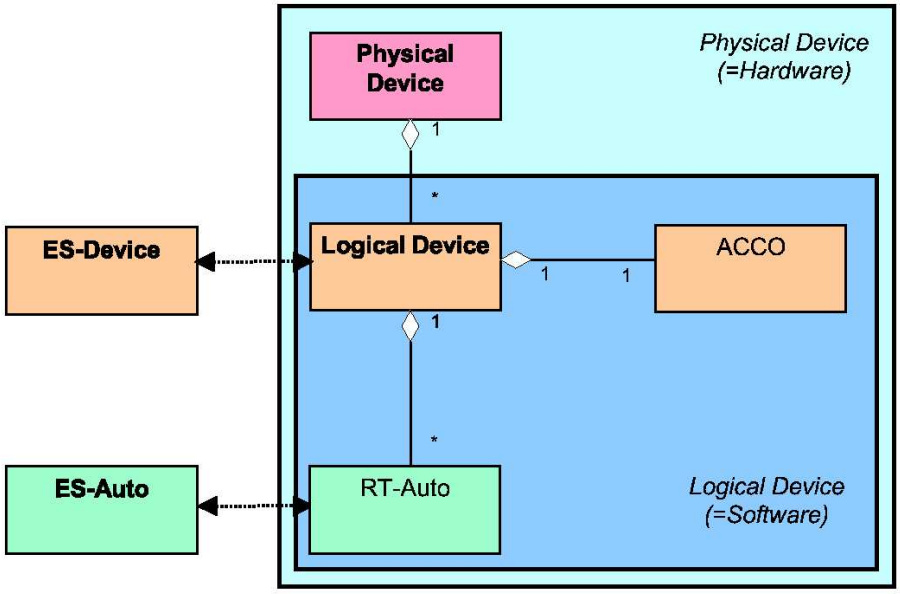
PDev obsahuje alespoň jedno **LogicalDevice** (označeno LDev), které je tvořeno softwarem (firmware), jako autonomní jednotka. Běžný poměr mezi PDev a LDev je 1:1, což znamená, že jeden softwarový program (nebo několik kooperativně pracujících programů) běží na jednom hardwaru. LDev nabízí běžné automatizační funkce, jako je identifikace nebo diagnostika zařízení. Slouží jako výchozí bod (objekt) pro přístup k RTAuto objektům.

ACCO (Active Control Connection Object) objekt implementuje konfigurovatelná propojení mezi **RTAuto** (Runtime Automation Object) objekty. RTAuto reprezentuje funkční výkonou část zařízení (vstupy/výstupy). Na jedno LDev zařízení připadá jeden ACCO objekt.

Každému RTAuto objektu v reálném zařízení odpovídá ES-Auto[2] objekt v návrhovém

[1] **D**istributed **C**ommon **O**bject **M**odel

[2] **E**ngineering **S**ystem, tzn. obraz reálného objektu v návrhovém prostředí



Obrázek 1.1: Runtime model

prostředí, LDev zařízení v době návrhu odpovídá ES-Device. Díky této reprezentaci může návrh celého systému probíhat bez znalostí implementace konkrétního zařízení, návrhář jen definuje propojení mezi jednotlivými objekty.

IP adresa PROFInet serveru jednoznačně identifikuje PDev zařízení, k LDev a k RTAuto objektům se přistupuje pomocí jednoznačného jména definovaného v rámci návrhu.

Logické zařízení se může nacházet v jednom z následujících stavů: **Non Existent** - zařízení není napájeno (neexistuje); **Initializing** - probíhá inicializace zařízení; **Ready** - zařízení je připraveno, výstupy RTAuto zařízení jsou v definované hodnotě; **Operating** - pracovní režim zařízení; **Defect** - nastala blíže nespecifikovaná chyba, je nutný vnější zásah.

1.3 COM terminologie

LDev, PDev a RTAuto je v PROFInetu představován COM[3] objektem, který se skládá ze dvou základních částí, které můžeme od sebe oddělit:

- interface (rozhraní) - zprostředkovává přístup k objektům, je jednoznačně deklarováno, není závislé na vývojovém prostředku.
- implementace - vlastní implementace deklarovaných metod rozhraní.

Pro specifikaci rozhraní COM objektu je použit *Interface Definition Language* (IDL jazyk), jehož syntaxe je velice podobná jazyku C. Definice rozhraní se skládá z několika prvků:

[3] **C**ommon **O**bject **M**odel

atributů rozhraní uvedených v hranatých závorkách, klíčového slova interface za kterým následuje název rozhraní a za znakem ":" jméno rodičovského rozhraní. Ve složených závorkách je obsažena deklarace metod rozhraní.

Metody rozhraní jsou definovány jako výstupní (uvozené klíčovým slovem *propget*) nebo vstupní (uvozené klíčovým slovem *propput*). Popis IDL jazykem je určen především klientům a proto se na popis obsažený v IDL souboru musíme dívat z pohledu klienta. Proto vstupní metoda, která provádí zápis do COM objektu je definována z pohledu klienta klíčovým slovem *propput*. Příklad definice rozhraní COM objektu je na obr. 1.2, více informací o IDL jazyku [5].

```
[
 object, uuid(4DDFB88B-AD0E-4D65-BDB7-8A0EDEB77501),
 dual, helpstring("Aqarium Up"),
 pointer_default(unique)
]

interface IAqa1 : IDispatch
{
  // výstupní metoda
    [propget] HRESULT pH([out, retval] long *pVal);
  // vstupní je současně definována i jako výstupní metoda
  // z důvodu možnosti čtení
    [propget] HRESULT kysliceni([out, retval] VARIANT_BOOL *pVal);
    [propput] HRESULT kysliceni([in] VARIANT_BOOL Val);
};
```

Obrázek 1.2: Definice rozhraní v jazyce IDL

Argumenty metod jsou nazývány *atributy*[4]. Metody umožňují serveru nabízet své služby (funkce) klientům.

Komunikace mezi serverem a klientem může probíhat synchronně nebo asynchronně. Asynchronní komunikace se používá pro zpětné informování serveru, že požadovaná funkce (metoda) na straně klienta skončila (např. ventil dosáhl polohy *zavřeno*). Díky asynchronní komunikaci odpadá komunikace typu *polling*, server nemusí cyklicky testovat stav klienta. Tento způsob komunikace se v PROFInetu označuje *event* - zasílání událostí.

1.3.1 Rozhraní IUnknown

Každé COM rozhraní je odvozeno od základního rozhraní, od tzv. *IUnknown* rozhraní. Obsahuje přesně tři metody: *AddRef()*, *Release()* a *QueryInterface()*.

COM definuje tato pravidla, která určují životnost (*lifetime*) objektu:

- pokud je vytvořena další reference na objekt, musí být zavolána metoda *AddRef()*

[4] v anglickém jazyku *property*

- pokud je rušena reference na objekt, musí být zavolána metoda *Release()*

Návratová hodnota je počet referencí na konkrétní objekt, pokud dosáhne nuly, objekt není užíván žádným jiným objektem a je odstraněn z paměti.

QueryInterface() - metoda pro dynamické procházení rozhraní, jako parametr ji předáme jméno rozhraní, na které chceme získat ukazatel, pokud toto rozhraní neexistuje, je navrácen chybový kód.

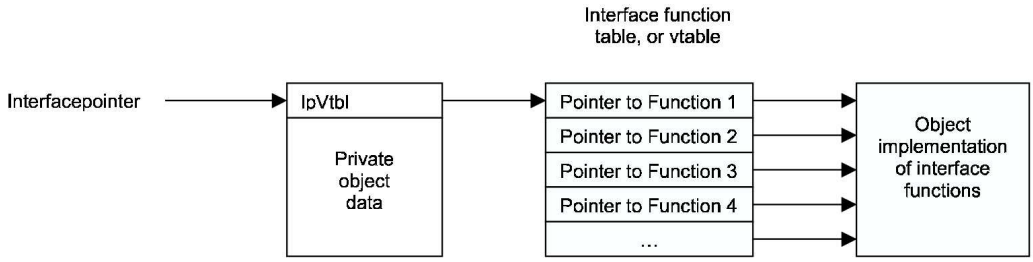
1.3.2 Rozhraní IDispatch

PROFInet objekty vedle povinného rozhraní IUknown implementují další standardní rozhraní *IDispatch*, které slouží k nepřímému volání metod objektu. Toto rozhraní definuje celkem čtyři metody, nejdůležitější metoda *invoke()* a *GetIDsOfName()*. Klientská aplikace zná pouze jméno volané funkce, které předá jako parametr funkci *GetIDsOfName()*. Funkce vyhledá v tabulce uložené v komponentě jaké jedinečné číslo *DispID* (v rámci jedné komponenty) odpovídá zadanému jménu funkce a vrátí jej klientovi. Se získaným *DispID* následně volá metodu *Invoke()*, která provede požadovanou funkci. Díky tomuto systému lze v programu volat funkce, které v době překladu nejsou známy[5]. Více informací [5].

COM model definuje tyto tři komunikační cesty:

- InProc - komunikace v rámci jednoho procesu
- InterProc - mezi dvěma procesy ale v rámci jednoho "počítače"
- Remote - mezi dvěma zařízeními (počítači) pomocí rozšířeného protokolu COM, nazývaného DCOM.

V PROFInetu je komunikace typu *InProc* převedena na přímé volání funkce pomocí tabulky virtuálních funkcí (obr. 1.3).



Obrázek 1.3: Volání metod v rámci jednoho procesu - InProc

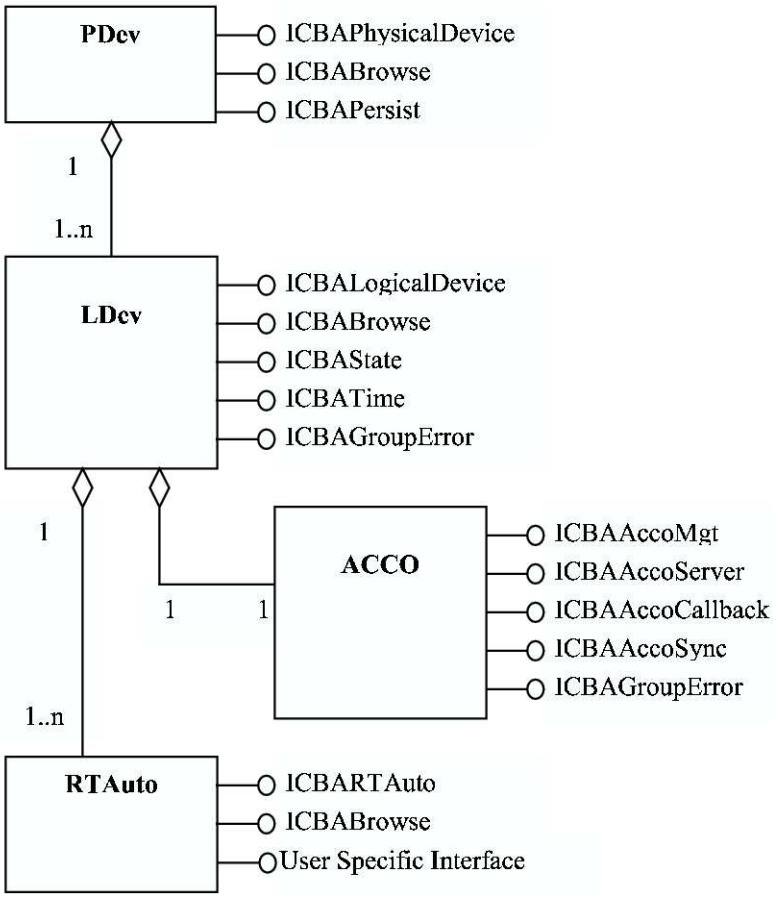
[5]tento způsob volání se označuje *OLE Automation*

1.4 Rozhraní PROFInetu

PDev, LDev a RTAuto objekty mají definováno rozhraní *ICBABrowse* pomocí něhož lze získat ukazatel na následující objekt v hierarchii (obr. 1.4). Nad objektem RTAuto je definováno tzv. *User Specific Interface*, kterým je dán prostor pro definici vlastních rozhraní (funkcí), která definují funkci celého serveru.

Každý objekt má základní rozhraní (*ICBAPhysicalDevice*, *ICBALogicalDevice...*), na které získáme ukazatel při přechodu z nadřazeného objektu pomocí rozhraní *ICBABrowse*. Toto rozhraní poskytuje základní informace o objektu, jako je: název, výrobce, seriové číslo, datum sestavení atp.

Protože popis jednotlivých rozhraní a pochopení jejich funkce je důležitý pro další práci, následuje jejich krátký popis. Více informací lze získat v [3]. Verze 2.0 PROFInet serveru rozšiřuje sadu rozhraní o další, viz [4].



Obrázek 1.4: Seznam rozhraní nad objekty PROFInetu

Physical Device

- *ICBAPhysicalDevice* - informace o hardwaru, výchozí bod pro objekty LDev.

- *ICBABrowse* - přechod na LDev zařízení.

Logical Device

- *ICBALogicalDevice* - informace o softwaru, výchozí bod pro objekty RTAuto a ACCO.
- *ICBABrowse* - přechod na objekty RTAuto
- *ICBAState* - čtení stavu LDev objektu.
- *ICBATime* - vlastní čas LDev objektu.
- *ICBAGroupError* - diagnostika zařízení (rozebrána v práci [1]).

RTAuto Objekt

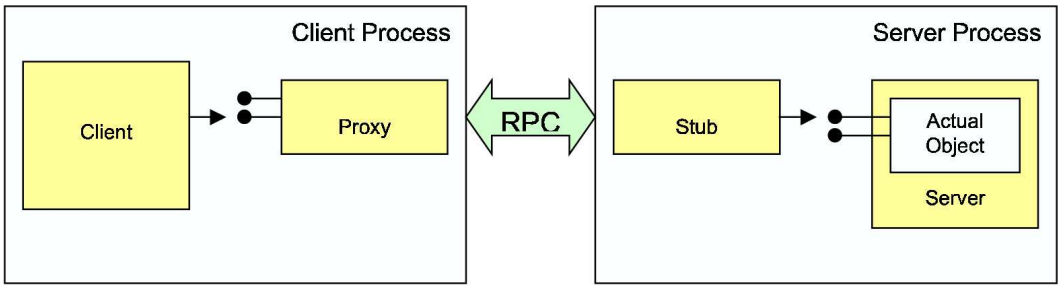
- *ICBARTAuto* - identifikace objektu.
- *ICBABrowse* - přechod na *User Specific Interface*.
- *User Specific Interface* - uživatelem definované rozhraní pracující nad atributy objektu.

ACCO Objekt

- *ICBAAccoMgt* - tvorba a rušení propojení mezi objekty.
- *ICBAAccoServer* - konfigurace poskytovatele dat.
- *ICBAAccoCallback* - spravuje asynchronní vyvolání události na straně příjemce dat.
- *ICBAAccoSync* - synchronní čtení/zápis atributů.
- *ICBAGroupError* - diagnostika (rozebrána v práci [1]).

1.5 Komunikace

Při komunikaci typu *InterProc* a *Remote Call* (obr. 1.5) je použit model volání proxy/stub (viz. [5]). Proxy se směrem ke klientovi chová jako originální volaný objekt, jako by byl na lokálním zařízení. Stub vykonává řádné volání objektu na straně serveru a návratovou hodnotu vrací přes RPC a proxy zpět klientovi. Spojení přes proxy/stub pomoci RPC obstarává COM, je zcela transparentní.



Obrázek 1.5: Volání metod mezi procesy - InterProc a Remote Call

1.5.1 ACCO objekt

ACCO objekt vytváří a udržuje propojení mezi RTAuto objekty. Tyto propojení jsou vytvářeny a rušeny dynamicky za běhu serveru, jsou nahrávány z konfiguračního a monitorovacího softwaru iMap.

Data jsou přenášena v definovaných časových intervalech (QoS[6]), společně s kvalitativní známkou (QC[7]). Objekty, mezi kterými probíhá přenos dat jsou označovány jako *provider* - zdroj dat a *consumer* - příjemce dat. Iniciátorem při vytváření spojení je zdroj dat, datové typy přenášených dat musí být stejné jak u poskytovatele tak u příjemce dat. Při načtení konfigurace rozhraní z IDL souboru musí mít pro poskytovatele dat atribut *[propget]*, pro příjemce dat *[propput]*.

Příjemce i poskytovatel si udržuje vlastní záznamy o vytvořených spojení. Vytvářet/rušit spojení lze pomocí rozhraní ACCO objektu *ICBAAccoMgt*, konfigurace již vytvořeného spojení (např. změna QoS) se mění pomocí rozhraní *ICBAAccoServer*. Rozhraní *ICBAAccoCallBack* obsahuje funkci, kterou poskytovatel volá na straně příjemce při žádosti o přenos hodnoty.

1.5.2 Marshalling

Marshalling provádí konverzi argumentů předávaných metodě do správného tvaru pro dané zařízení. Jedná se o konverze typu little/big endian, kódování čísel apod. Dále se stará o vytváření *proxy* na straně klienta a *stub* na straně serveru pro volané metody. Jejich strukturu zná z popisu rozhraní (metod) IDL jazykem.
Existují dva druhy marshallingu:

- **universal marshalling** - používán v Javě, Visual Basicu. Pro volání metod definované v rozhraní používá IDispatch rozhraní.

- **standard marshalling** - používán v C/C++.

V PROFInetu je použit standardní marshalling, který využívá standardu RPC. Rozhraní objektů PDev, LDev a RTAuto jsou definovany jako OLE Automation - přístup je možný

[6] **Q**uality **o**f **S**ervice
[7] **Q**uality **C**ode

pomocí IDispatch interface, narozdíl od objektu ACCO[8], ke kterému se může přistupovat jen při použití standardního marshallingu - nemá rozhraní IDispatch. Aby mohly jazyky jako je Visual Basic k ACCO objektu přistupovat je nutné použít tzv. *Automation Wrapper*, který Visual Basic aplikaci poskytne IDispatch rozhraní.

Způsob konverze argumentů je podrobněji popsán v kap. 2.6. Více informací lze získat v [3] a [2].

1.6 Shrnutí

Popsané vlastnosti PROFInet serveru jsou součástí specifikace, ale v použité verzi PROFInet serveru 1.2 nejsou zdaleka všechny implementovány. Několik zásadních omezení této verze:

- možný poměr mezi LDev a RTAuto objektem je pouze 1:1.
- není podporována komunikace typu *zasílání zpráv*.

[8] nemá OLE Automation rozhraní, díky omezené množině datových typů, které tento typ marshallingu umožňuje

Kapitola 2

PROFInet a Linux

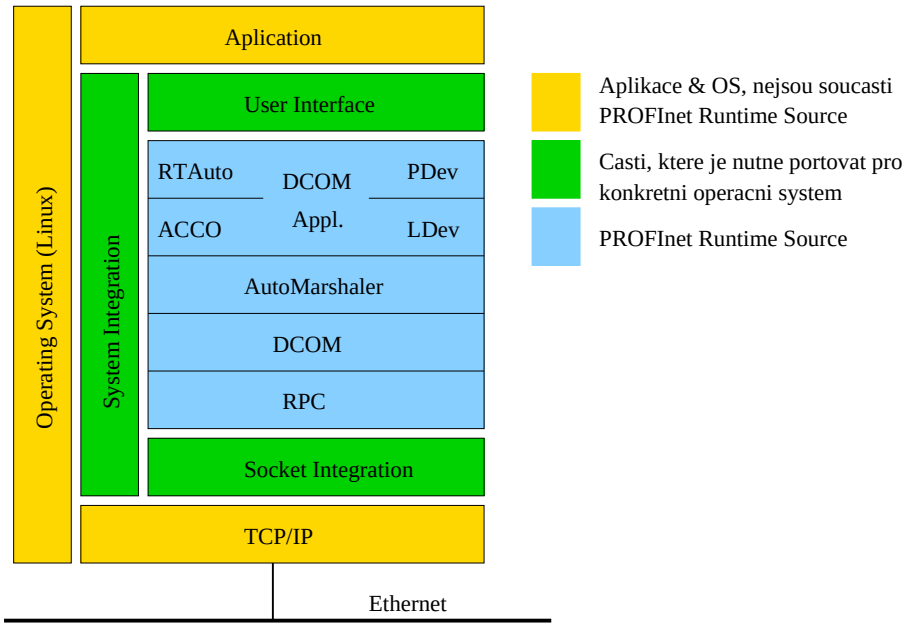
V této kapitole lze nalézt informace o adaptaci PROFInet serveru pod operační systém Linux. Je zde nastíněn princip funkce jednotlivých částí serveru (base, rpc, dcomrt, dcomap) a jejich vzájemná spolupráce. Tato fakta se opírají převážně o dokumentaci [2] a vlastní zkušenost s adaptací.

2.1 Úvod

Operační systém Linux je již časem prověřený, stabilní operační systém, který oproti jiným operačním systémům je volně šiřitelný pod licencí GPL[1]. Monolitické jádro operačního systému, které odděluje hardware počítače od aplikací, je velice dobře škálovatelné. Díky tomu lze Linux provozovat i na počítačích s velice omezenými systémovými zdroji. PROFInet lze obecně převést do libovolného operačního systému, který má ANSI C překladač, umožňuje běh více vláknových aplikací (multi-threading) a má implementovánu TCP/IP vrstvu (obr. 2.1). Vrstva TCP-IP není součástí PROFInet implementace.

Přístup k TCP-IP vrstvě je v Linuxu možný pomocí *BSD socket interface*, který je součástí knihovny Glibc. Jakožto každý Unixový systém podporuje multi-threading (knihovna pthread). Jako překladač ANSI C jazyka lze použít *gcc* [8].

Zdrojový kód PROFInet serveru lze získat z web stránek společnosti PROFIBUS International http://www.profibus.com[2], tato část kódu je na obr. 2.1 zvýrazněna modře. Část označena zeleně, je specifická pro konkrétní platformu, pro konkrétní operační systém. Primárně je PROFInet server vyvíjen ve verzi pro operační systém VxWorks a Windows, ve které je také dostupný na web stránkách firmy PROFIBUS. Verze pro operační systém Linux se objevila až nedávno.



Obrázek 2.1: Struktura PROFInet runtime software

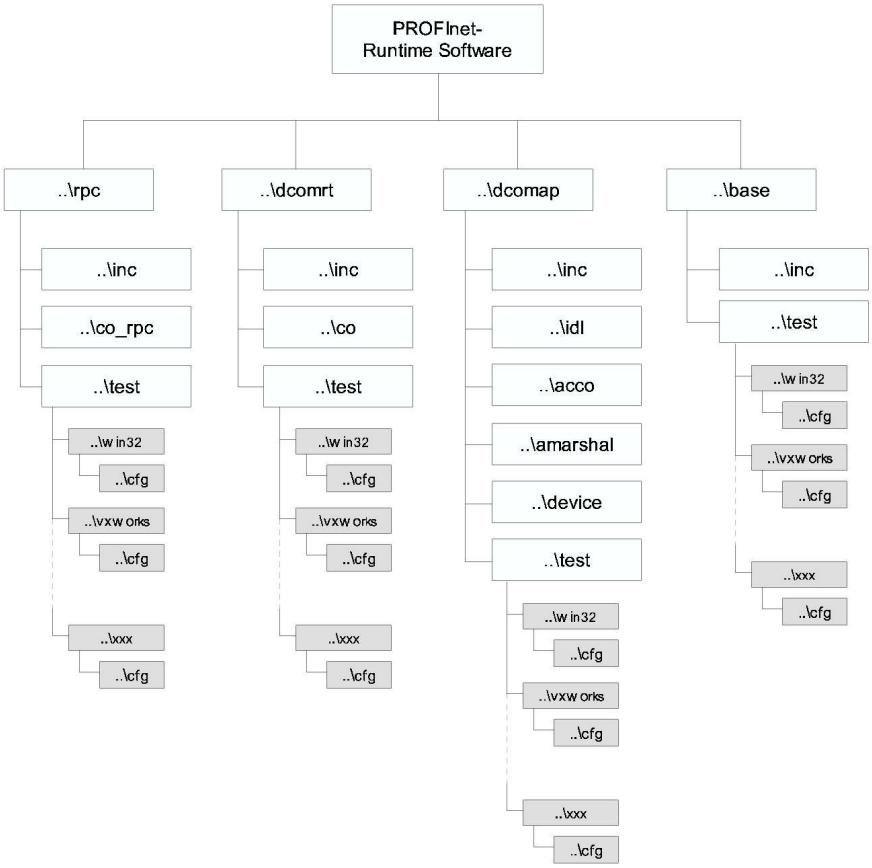
2.2 Struktura PROFInet serveru

Na obr. 2.1 je znázorněno, které komponenty[3] PROFInet serveru musí být adaptovány v závislosti na cílové platformě. Tyto části jsou od jádra PROFInetu odděleny, jsou ve zvláštních adresářích, pojmenovaných podle cílové platformy (obr. 2.2).

Jednotlivé komponenty PROFInet serveru jsou od sebe odděleny podle funkcí, které vykonávají (detailněji popsány v následujícím textu):

- **base** - trasovací systém PROFInetu, společné funkce všem ostatním částem.
- **rpc** - implementace RPC[4] systému.
- **dcomrt** - implementace DCOM vrstvy.
- **automarshall** - typové konverze.
- **acco** - realizuje a udržuje propojení mezí objekty.
- **device** - podpůrné funkce pro vytvoření modelu PROFInetu (PDev, LDev a RTAuto objekty).

[1]**G**eneral **P**ublic **L**icence
[2]přístupné jen pro členy organizace
[3]výraz *komponenta* je v PROFInet terminologii používáno jak pro označení jednotlivých částí serveru (RPC, BASE, DCOM, Marshaller) tak i pro celý PROFInet server jako dílčí část větší technologie.
[4]**R**emote **P**rocedure **C**all



Obrázek 2.2: Struktura adresářů

Každá z komponent má svoji vlastní adaptaci do systému, na kterém bude provozována nezávisle na ostatních komponentách (šedé adresáře na obr. 2.2). Úpravy prováděné programátorem při převodu softwaru na jinou platformu jsou povoleny pouze v těchto adresářích.

2.3 Komponenta BASE

Jsou zde definována především makra pro práci s kritickými sekcemi, s alokací paměti, s řetězci. Dále je zde implementován *trasovací* systém, který se z ostatních vrstev volá pomocí maker, podle charakteru žádaného výpisu a důležitosti.

2.3.1 Kritické sekce

Pro práci s kritickými sekcemi byly zvoleny tzv. *mutexy* standartu POSIX [10], [9], ty umožňují uzamknout objekt, k němuž pak má přístup jen jedno vlákno. Další alternativou je použití semaforů.

Pro práci s mutexy jsou definovány následující funkce, každá z komponent PROFInetu si zavádí svá pomocná makra, která jsou směřována na tyto funkce. Argumentem těchto funkcí je datová struktura, která má prvky typu: **pthread_mutex_t** (vlastní mutex) a **pthread_t** (vlákno, které vlastní mutex - v současné verzi PROFInetu (v1.2) nepoužito).

InicializeCriticalSection(pCritSec) - inicializuje mutex.
DeleteCriticalSection(pCritSec) - Ruší mutex.
EnterCriticalSection(pCritSec) - Zamkne (mutex) kritickou sekci pro ostatní procesy, ostatní procesy musí čekat na uvolnění.
LeaveCriticalSection(pCritSec) - Odemčení kritické sekce.

2.3.2 Trasovací systém

Pro každou komponentu serveru lze nastavit jednu z úrovní trasovacích zpráv. Jsou definovány tyto úrovně (uspořádáno sestupně) trasovacích zpráv a chybových hlášení: FATAL, ERROR, UNEXP, WARN, NOTE, CHAT. Při zpravě typu FATAL je vážně narušen chod systému, nelze pokračovat v činnosti. Naopak zpráva typu CHAT je informativní a má spíše význam pro ladění systému.

Každá komponenta PROFInetu vkládá hlavičkový soubor podle svého jména (*xxxtrc.h*, kde xxx je jméno komponenty). Každý soubor komponenty má definováno specifické makro (obr. 2.3), podle kterého se v *xxxtrc.h* určí, o který soubor se jedná. Nastaví se makra:

CBA_TRACE_FILE_ID a CBA_TRACE_FILE_NAME.

Dále se nastaví úroveň trasování celé komponenty makrem CBA_TRACE_LEVEL a nastaví se pomocné makro CBA_COMPONENT_XXX pro hlavičkový soubor *gtrace.h*.

Do souboru *xxxrc.h* se vkládá další soubor *gtrace.h* ve kterém se nadefinuje podle jména komponenty makra

CBA_TRACE_COMPONENT_NAME a CBA_TRACE_COMPONENT_ID

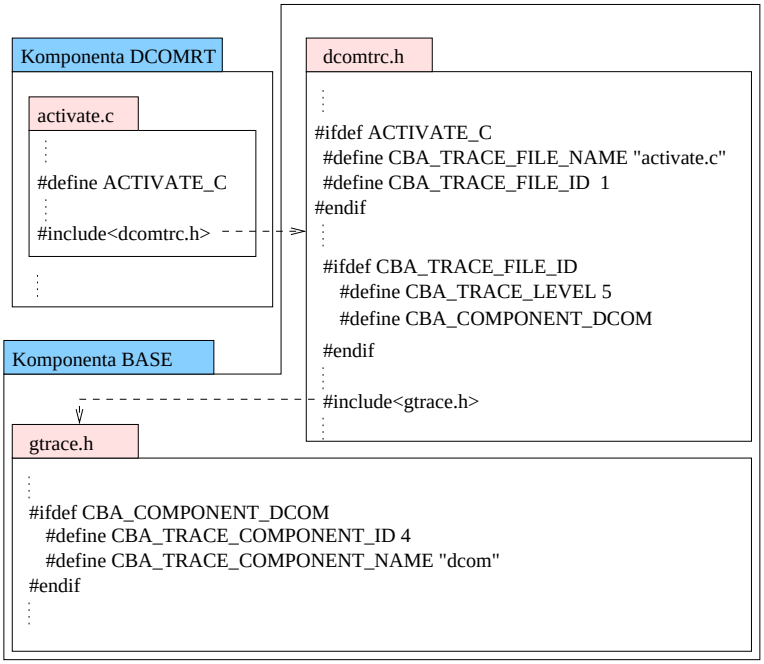
společné všem souborům komponenty.

Díky tomuto sytému lze snadno v trasovacích výpisech uvádět název komponenty, jméno souboru odkud zpráva pochází a volit důležitost trasovacích výpisů pro každou z komponent.

Na obr. 2.3 je příklad definice úrovně trasovacích hlášení pro komponentu DCOM (v hierarchii adresářů označena jako DCOMRT). Soubor *activate.c* nastaví specifické makro a vloží do své hlavičky soubor *dcomtrc.h* který vkládají i ostatní zdrojové soubory komponenty. Dojde k definici pravého jména souboru, které se využije v trasovacích hlášeních a nastaví se trasovací úroveň, dále se vloží soubor *gtrace.h*, kde se nastaví jméno komponenty.

Trasovací výpis se volá makrem: *CBA_TRACE_X()*, kde X je počet argumentů. Pokud je X rovné nule, má funkce argumenty: *LEVEL* - udávající uroveň této hlášky a *MESSAGE* - zpráva, která se zobrazí. Číslo X udává počet dalších argumentů, které se vytisknou ve zprávě (stejná syntaxe jako u příkazu *printf()*).

Funkce definované v této komponentě tvoří základ pro ostatní komponenty PROFInetu. Pro bližší informace doporučuji nahlédnout do zdrojového kódu této komponenty (konkrétní soubory uvedeny níže).



Obrázek 2.3: Princip funkce trasovacího systému

Soubory určené pro adaptaci	
./v1.2-src/base/test/linux/cfg	*trccfg.h*
	trccfg.c
	basecfg.h

2.4 Komponenta RPC

Na programátorovi je napojit implementovanou TCP/IP vrstvu operačního systému na RPC[5] vrstvu PROFInetu. Stavy vyvolané na TCP/IP vrstvě jsou do RPC vrstvy předávány pomocí metod:

- OnConnectInd() - indikace žádosti o nové připojení.
- OnConnectCnf() - připojení vytvořeno.
- OnTCPDataInd() - indikace dat na soketu.
- OnTCPClose() - klient uzavřel spojení.

RPC s TCP/IP vrstvou komunikuje pomocí metod:

- RPC_CONNECT() - žádost o vytvoření nového komunikačního soketu.
- RPC_SEND() - poslání dat.

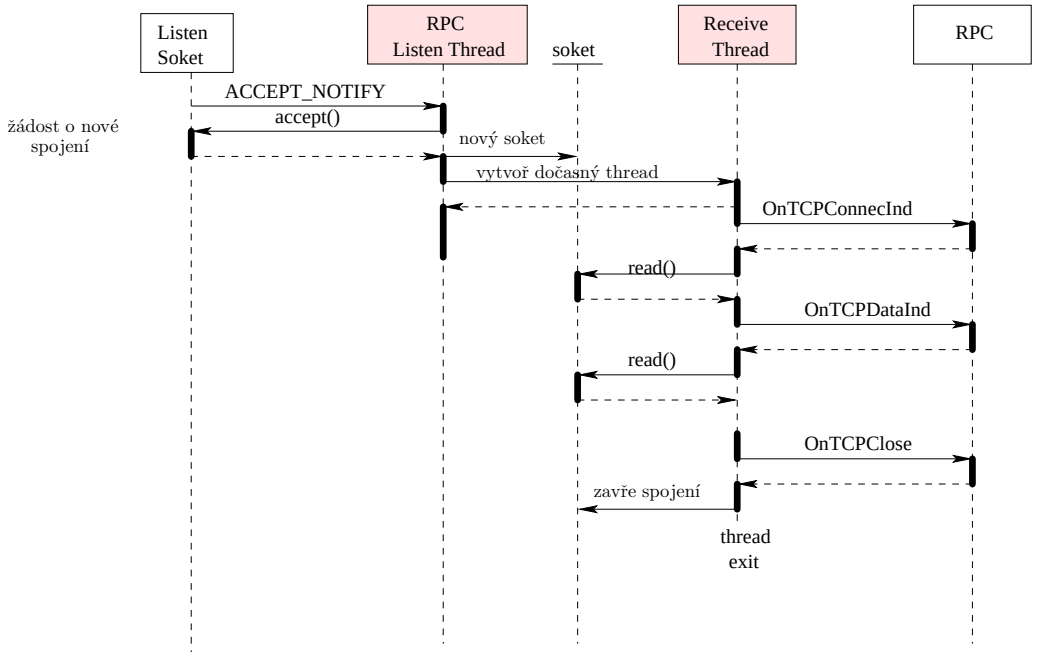
[5]**R**emote **P**rocedure **C**all

- RPC_CLOSESOCKET() - uzavření soketu.

dále je nutné implementovat komplexnější metody, nejdůležitější z nich:

- RpcSysTCP_Listen() - vytvoří *listen* soket, na kterém server přijímá požadavky a proces *Listen_Thread*, který ho obsluhuje.

- RpcSysTCP_Unlisten() - zavře *listen* soket.



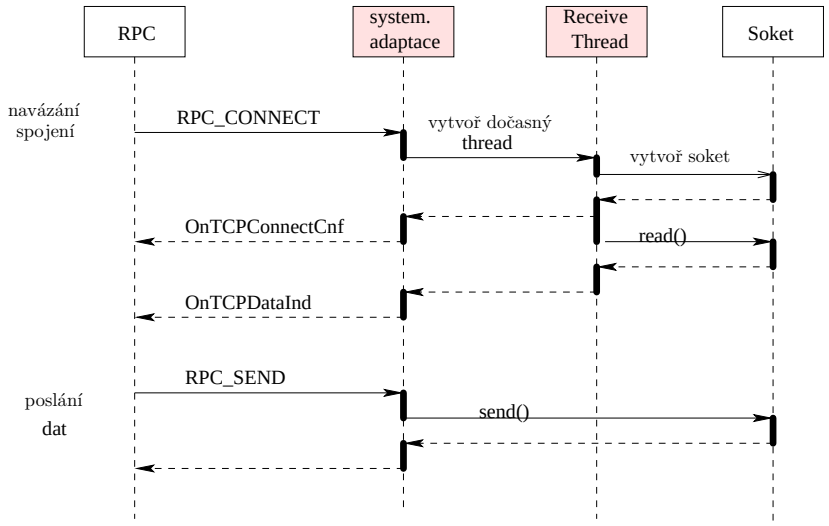
Obrázek 2.4: Žádost o nové připojení

Příchozí spojení jsou obsluhovány samostatným procesem *Listen_Thread()* (obr. 2.4)[6], který je spuštěn při startu serveru metodou *RpcSysCreateActiveSocket()*. Při žádosti o nové připojení vytvoří *Listen_Thread()* další proces *Receive_thread()*, který přečte data z příchozího spojení a skončí. Při vytvoření tohoto vlákna je vrstva RPC informována voláním metody *OnTCPConnectInd()* o novém spojení a metodou *OnTCPDataInd()* o datech na konkretním soketu obsluhovaným příslušným *Receive_thread()*.

Spojení je ukončeno vzdáleným klientem, který žádal o připojení nebo RPC vrstvou, metodou *OnTCPClose()*.

Pokud chce server poslat data klientovi, vytvoří metoda *RPC_CONNECT()* aktivní soket a proces (*ReceiveThread()*), který tento soket obsluhuje. Úspěšné vytvoření tohoto soketu je předáno RPC vrstvě metodou *OnTCPConnectCnf()*. Data jsou posílána metodou *RPC_SEND()* (obr. 2.5).

[6] část, která se musí adaptovat podle cílové platformy je na následujících obrázcích zvýrazněna *růžově*



Obrázek 2.5: Aktivní připojení

Soubory určené pro adaptaci	
./v1.2-src/rpc/test/linux/	*rpc_cfg.h*
	rpc_cfg.c

2.5 Komponenta DCOM

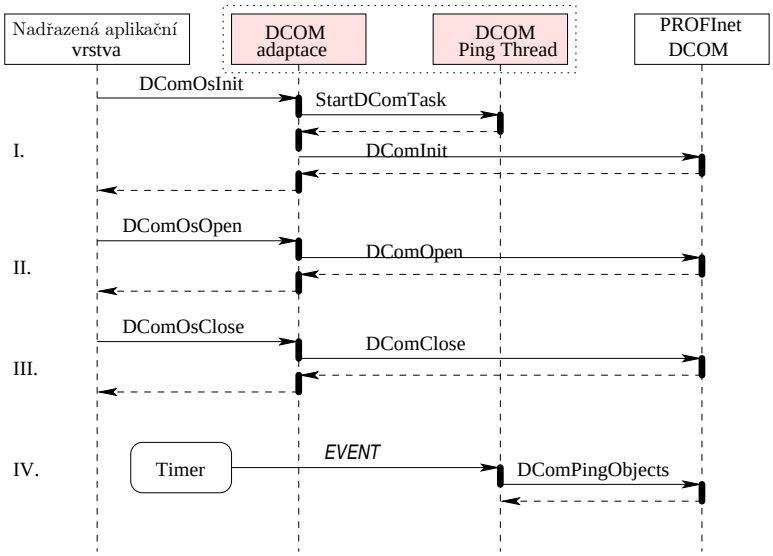
PROFInet server je navržen tak, aby mohl být implementován i pod systémy, kde není dostupný DCOM, proto přináší vlastní implementaci vrstvy DCOM. Na systémech, kde se DCOM nachází, musí být vypnut, aby byl dostupný standartně definovaný port pro DCOM, port 135. Na Linuxovém systému musí být PROFInet server spuštěn s právy uživatele root, protože běžný uživatel nemůže otevřít port s číslem nižším jak 1024.

Budoucí verze PROFInet serveru (v2.0) by měla umět využívat implementovaný DCOM v systémech MS Windows 2000, PROFInet server se bude chovat jako standardní COM aplikace.

Programátor musí adaptovat několik funkcí (sekvencí) DCOM vrstvy (obr. 2.6):

- Inicializace DCOM vrstvy (sekv. I, obr. 2.6). Vytvoří propojení s nižší RPC vrstvou.
- Spuštění DCOM (sekv. II, obr. 2.6). Vytvoří server soket na portu 135.
- Zavření DCOM (sekv. III, obr. 2.6). Zastaví *DComPingThread* a zavolá metodu *DComClose* vrstvy DCOM.
- Implementování časovače (Timer) (sekv. IV, obr. 2.6) pro cyklickou kontrolu dostupnosti objektů. Timer řešen jako samostatný proces *DComPingThread*, který periodicky volá funkci *DComPingObject*.

Funkce jsou volány z nadřazené aplikační vrstvy (přímo z hlavní *main()* funkce) pomocí metod *DComOsInit()*, *DComOsOpen()* a *DComOsClose()*.



Obrázek 2.6: Sekvence DCOM vrstvy

Soubory určené pro adaptaci	
./v1.2-src/dcomrt/test/linux/cfg	*dcomcfg.h*
	dcomcfg.c

2.6 Komponenta Automarshal

Automarshaller vytváří proxy a stub rozhraní pro všechny metody popsané IDL souborem. Kód pro proxy a stub je z IDL souboru generován MIDL[7] kompilátorem a upraven programem TypeLib converter (je součástí profinetu), pak je staticky zahrnut do kódu serveru.

2.6.1 Inline assembler

Převážná část kódu určená k adaptaci je v inline[8] asembleru. Linuxový inline asembler používá AT&T syntaxi [6], která se od *intelovského* asembleru používaného v systému Windows především odlišuje v těchto bodech:

- pořadím parametrů (nejprve zdroj, pak cíl).
- konstantní parametry uvozeny znakem $.
- nutností udávat datový typ parametrů.
- syntaxe adresování paměti je zcela odlišná.

[7] **M**icrosoft **IDL** compiler
[8] asembler, který je přímo vkládán do zdrojového kódu jazyka C

Na obrázku 2.8 je příklad nejprve v assembleru itelovském a pak v asembleru AT&T. Samotná syntaxe assembleru AT&T je naznačena na obrázku 2.7.

```
__asm__ __volatile__ ("assembler kód"
  : mapování výstupních proměnných
  : mapování vstupních proměnných
  : seznam modifikovaných registrů, seznam modifikované paměti );
```

Obrázek 2.7: Syntaxe AT&T assembleru

Klíčové slovo *__volatile__* říká překladači, aby námi zapsaný kód v assembleru dále neoptimalizoval. Na proměnné použité v jazyku C se v inline assembleru odkážeme pomocí zápisu **%x**, kde **x** určuje pořadí proměnné v seznamu mapovaných proměnných v assembleru (obr. 2.7). Znak %% uvozuje jména registrů. V zápisu mapovaných proměnných lze použít dalších atributů (v našem příkladě *"g"*), které slouží zejména k optimalizaci výsledného kódu překladačem.

```
Windows:   asm {mov edx, size};
Linux:     __asm__ __volatile__ ("movl %0, %%edx": :"g"(size):"%edx");
```

Obrázek 2.8: příklad inline assembleru v systému Windows a Linux

Důvodem použití tak značné části kódu v assembleru je, že potřebujeme mít úplnou kontrolu nad voláním funkcí, pomocí asembleru nahradíme část, kterou za nás jinak dělá překladač jazyka C utomaticky (tj. překopírování parametrů metody na zásobník, zavolání metody, vrácení návratové hodnoty funkce).

2.6.2 Konfigurace

Automarshaller lze konfigurovat následujícími makry, jejichž nastavení musí korespondovat s možnostmi cílového systému:

- AM_MAX_ARGUMENTS - definuje maximální počet argumentů metod, implicitně nastaven na 16.
- AM_MAX_COMPONENTS - maximální počet prvků v datové struktuře.
- AM_USE_FLOATINPOINT - zapnout podporu čísel s pohyblivou desetinou čárkou, implicitně povoleno.
- AM_USE_INT64 - podpora 64-bitového datového typu integer, implicitně zakázáno.
- AM_CLIENT_HEAP_SIZE, AM_SERVER_HEAP_SIZE - velikost heap paměti pro marshalling, implicitně 1kb.

název v marshalleru	interpretace	dat. typ v Linux
BYTE	kladné 8bit číslo	unsigned char
WORD	kladné 16bit číslo	unsigned short
DWORD	kladné 32bit číslo	unsigned int
FLOAT	desetiné číslo	float
BOOL	logický typ, 16bit číslo	unsigned short
UINT	kladné 32bit číslo	unsigned int
QWORD	64bit	pomocná struktura

Tabulka 2.1: Datové typy

Dalším krokem je sjednocení datových typů (tab. 2.1) mezi různými platformami. Datový typ QWORD (64bit) je v operačním systému Linux implementován jako struktura s dvěma prvky - vyšší a nižší DWORD (dvě samostatné proměnné o 32bitech).

Při volání metody deklarované pomocí IDL jazyka:

```
HRESULT LogicalDevice([in]          BSTR                Name,
                      [out, retval] ICBALogicalDevice **ppLDev);
```

je stav na zásobníku naznačen v tab. 2.2. Automarshaler ukládá parametry metody na zásobník následujícím způsobem:

- **proměnná o velikosti do 4 byte** - proměnné typu signed/unsigned char, short, long, float a ukazatele jsou uloženy jako typ DWORD (double word).
- **proměnná o velikosti 8 byte** - proměnné typu double a date jsou uloženy jako typ dvou DWORD.
- **datová struktura** - ukládá se jen pointer na tuto strukturu.

adr. na zásob.	jméno parametru	uložený typ
zásobník	ICBALogicalDevice **pLDev	DWORD - ukazatel LDev
zásobník - 1	BSTR Name	DWORD - ukazatel na Name
zásobník - 2	ukazatel na rozhraní	DWORD
zásobník - 3	návratová adresa metody	DWORD

Tabulka 2.2: Stav zásobníku při volání metody

Adaptace funkcí a maker pomocí inline assembleru:

- AM_InterlockedIncrement - provádí inkrementaci proměnné, přičemž je zaručena exkluzivita přístupu, opakem je funkce AM_InterlockedDecrement.
- FORMAT_IID - provádí formátování identifikátoru rozhraní do srozumitelné textové podoby.

- AM_DEFINE_PARBUFFER(par, nArguments) - provádí alokaci pole typu DWORD o velikosti *nArguments*, vrací pointer *par* ukazující na jeho začátek.

- AM_UnMarshalParam - vyplňuje tabulku 2.2, tuto funkci není nutno adaptovat, uvedena jen pro úplnost.

- ASM_CALL_VOID_FUNC_PAR(func, par1, parBuffer) - provádí volání funkce určené ukazatelem *func* s polem argumentů na které ukazuje ukazatel *parBuffer*[9], na první z argumentů metody ukazuje *par1*.

Proces Marshallingu je skončen vytvořením pole (tab. 2.2) o maximální velikosti AM_MAX_ARGUMENTS typu DWORD s hodnotami (s ukazateli), které obsahují argumenty metody, ukazatel na rozhraní ke kterému metoda přísluší a návratové adresy.

Více informací lze získat v [2] a přímým studiem zdrojových souborů marshalleru.

Soubory určené pro adaptaci	
./v1.2-src/dcomap/test/linux/cfg	*am_cfg.h*
	am_cfg.c

2.7 ACCO objekt

Acco objekt, který vytváří, ruší a udržuje propojení mezi RTAuto objekty je snadno adaptován do Linuxu. Je zde jen několik časových konstant, které musí být konfigurovány (v souboru accocfg.h) jako je minimální hodnota pro QoS, časové meze pro rušení spojení atp. V [2] je doporučeno zachovat přednastavené hodnoty.

Jediná funkce k adaptaci je *CBA_Acco_ReviseQos()* která provádí revizi hodnoty QoS s ohledem na minimální hodnotu QoS (definována makrem), aby nedocházelo ke spojením s hodnotou QoS = 0. Zbytek funkcí není třeba adaptovat, jejich význam je podrobně rozebrán v [2].

K ACCO objektu patří i tak zvaný *perzistentní* proces, který běží na pozadí systému a ukládá aktuální informace o objektech (o propojení mezi objekty). Při obnovování běhu systému po jeho pádu může být tato konfigurace nahrána a systém nemusí být znovu konfigurován z návrhářské stanice (programem iMap).

Soubory určené pro adaptaci	
./v1.2-src/dcomap/test/linux/cfg	*accocfg.h*
	accocfg.c

[9] podle tab. 2.2 je na prvním místě pole argumentů poslední argument

2.8 Device

Tato část adaptace obsahuje podpůrné funkce pro vybudování modelu PROFInetu (viz dále). Je zde vytvořen obecný prostředek pro práci s dynamickými seznamy objektů, pro jejich vyhledávání, pro udržení jejich konzistence při současném přístupu několika procesů.

Mezi funkce, které je třeba adaptovat, patří: funkce pro práci s časem, datem a zejména funkce hlavního vlákna, která periodicky volá *CBA_Timer_Call* pro práci s uživatelskými časovači. Funkce pracující se seznamem objektů jsou zcela platformě nezávislé, způsob jejich použití je rozebrán v [2].

PROFInet používá své softwarové časovače, funkce CBA_Timer_Call pracuje nad uspořádaným seznamem časovačů (uspořádaný podle času vypršení), když některý z nich vyprší, volá se tzv. *Callback* funkce, která je s daným časovačem svázána. Časovače mohou být typu: **jednorázové** - Callback funkce je volána jednou po uplynutí času; **cyklické** - Callback funkce je volána periodicky s periodou časovače; **n-násobné** - Callback funkce je n-krát opakována s periodou časovače. Informace o struktuře časovačů lze nalézt v souboru ./dcomap/device/timers.c, popis časovačů v [2] je velice strohý.

Soubory vytvářející "strukturu" PROFInetu jsou zejména v adresáři ./dcomap/device. Jedná se o objektovou strukturu, kde základním prvkem je PDev objekt, na něj je navázán seznam s LDev objekty a seznam tabulky s atributy RTAuto objektů. Pro ilustraci je přiložena příloha 6, 7, 8.

Soubory určené pro adaptaci	
./v1.2-src/dcomap/test/linux/cfg	*DevCfg.h*
	DevCfg.c

2.9 Aplikace PROFInet

Jedná se o aplikaci vytvořenou nad již popsanými vrstvami. Při kompilaci se ke zdrojovému kódu PROFInetu přidá výstup po kompilaci IDL souborů popisující rozhraní, které je načteno do datové struktury. Z té je vytvořena hierarchická struktura objektů popsaná výše. Funkce pracující nad touto datovou strukturou se nalézají v souboru ./dcomap/test/linux/sysobj.c.

IDL soubor se nejprve kompiluje *MIDL* kompilátorem (součástí Visual Studia) a posléze se *AMCvtTlb* konvertorem převede do vhodného formátu (tento formát je důkladně popsán v [2]) pro začlenění do PROFInetu.

Při inicializaci PROFInet objektů je nejprve zavolána funkce *SysLinux_Startup()*, která vytvoří PDev objekt s jedním obecným LDev objektem. Funkce *SysLinux_Init-UserObjects()* načítá datovou strukturu vytvořenou AMCvtTlb konvertorem a podle ní vytváří funkcí *SysLinux_Add_LDev()* LDev objekty a RTAuto objekty. Posloupnost příkazů, jenž tato funkce provádí je naznačena na (obr. 2.9).

Tyto funkce není třeba adaptovat, jedná se o funkce, které volají již dříve implementované a adaptované funkce (např. obr. 2.9). Součástí PROFInet aplikace je i definice vlastní funkčnosti komponenty. Postup vytváření PROFInet komponenty je do podrobna popsán v [1] a v [2]. Detailní informace lze také získat ze zdrojového kódu.

```
SysLinux_Add_LDev() {
...
// Vytvoř nové LDev zařízení a registruj ho u PDev objektu
CBA_LDev_Construct();
...
 //přes všechny RTAuto objekty naležející k LDev
 while(otherRTAuto) {
   ...
   // volání funkce:
   SysLinux_Add_RTAuto() {
     // Vytvoř nový RTAuto objekt a registruj ho u LDev objektu
     CBA_RTAuto_Construct();
     ...
     // Registruj RTAuto objekt pro použití pomocí DCOM vrstvy
     CBA_RTAuto_Register();
   }
 ...
 }
...
// Registruj LDev objekt pro použití pomoci DCOM vrstvy
CBA_LDev_Register();
...
}
```

Obrázek 2.9: Postup vytváření PROFInet objektů

Soubory určené pro adaptaci	
./v1.2-src/dcomap/test/linux/	*sysobj.h*
	sysobj.c
./v1.2-src/dcomap/test/linux/cfg	*syscfg.h*
	syscfg.c

2.10 Překlad PROFInet serveru

Pro sestavení celé aplikace byl napsán *Makefile* [9], který se nachází v adresáři *./dcomap/test/linux*. Příkaz *make*[10] lze spustit s těmito argumenty (ale lze i vyvolat *make* pro konkretní soubor, např. *make trccfg.o*): **release**, **debug** a **clean**.

Argument **release** je implicitní, je generován čistý kód bez informací pro debuger[11]. Debug (*make* s argumentem **debug**) informace se skládají i se zdrojového kódu samotné aplikace, výsledný kód je nadměrně veliký. Příkaz *make* **clean** odstraní všechny **.o* soubory.

Jednotlivé zdrojové soubory se kompilují postupně, výsledek kompilace je uložen do

[10] interpretuje Makefile v aktuálním adresáři
[11] program pro trasování běhu aplikace

adresáře *./dcomap/test/linux*. Server byl odladěn pomocí kompilátoru *gcc* [8] verze 2.95.3.

2.11 Shrnutí

V této kapitole jsem nastínil postup adaptace PROFInet serveru z platformy s operačním systémem Windows na systém Linux (intelovská 32bit architektura). Díky členění aplikace do adresářů podle funkce a udržení odstupu obecného kódu od implementačně závislé je vše přehledné a čitelné. Kód PROFInetu koresponduje s dodávanou dokumentací [2]. Trasovací systém spolu s filtrem výpisu pro jednotlivé komponenty je podrobný a velice užitečný při ladění celého systému, každé volání i zdánlivě nepodstatné funkce může být zobrazeno v trasovacích výpisech.

Kapitola 3

Webové rozhraní

Tato kapitola vychází ze specifikace webového rozhraní popsaného v [4], některé části byli rozšířeny pro dosažení větší flexibility (v textu je na ně upozorněno). Tato specifikace je v tzv. *draft* (vývojové) verzi a proto se dá očekávat, že se bude časem měnit a dále rozšiřovat. V závěru kapitoly je popsána vlastní implementace tohoto rozhraní, která pracuje na Linuxovské platformě s webovým serverem Apache.

3.1 Úvod

Webové rozhraní pro PROFInet umožňují přístup ke komponentám PROFInetu pomocí standardních protokolů HTTP. Data jsou posílána ve standardizovaném tvaru (HTML, XML) a jsou zobrazeny obvyklým web prohlížečem (Microsoft Explorer, Mozilla, Opera...). Protokol DCOM není internetovým protokolem[1], port 135 není přístupný z okolního světa, proto přístup k PROFInetu přes standardní web prohlížeč je velice žádaný.

3.2 Cíl implementace

Specifikace popsaná v [4] definuje vlastnosti, které by měl PROFInet server s Webovým rozhraním splňovat:

- Adresování - jednoznačné adresování objektů a atributů PROFInetu.
- Design HTML stránek - definuje CSS styl stránek, rozvržení stránky atp[2].
- Názvy neměnných atributů PROFInet objektů, jako je *ConnState*, *ConnChannel*, atp.
- Zabezpečení - omezení přístupu k některým atributům komponenty.

Podle "množství" implementovaných vlastností lze server rozdělit do několika tříd, jejich popis udává tab. 3.1.

[1] bezpečnostní důvody

[2] ve specifikaci [4] není zcela dokončeno

třída	vlastnosti	role
třída 0	statické HTML stránky	povinné
třída 1	zahrnuje třídu 0 + dynamicky generované stránky	nepovinné
třída 2	zahrnuje třídu 1 + dynamické stránky v XML(+XSL) formátu	nepovinné
třída 3	zahrnuje třídu 2 + Webové služby	doporučené

Tabulka 3.1: Rozdělení PROFInet serveru podle hloubky web integrace

Třída 0 nabízí jen základní funkce. Systém této třídy nemůže být zaintegrován do složitějších celků, jako je topologie s tzv. *plant* serverem (viz. kap. 3.3). Stránky jsou staticky zaintegrovány v systému. Přístup je možný jen přes web prohlížeč.

Třída 1 a 2 nabízí větší flexibilitu, generování HTML stránek je dynamické, data získávána přímo z PROFInet serveru.

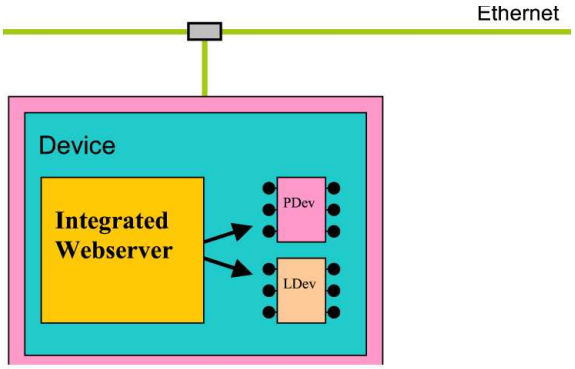
Třída 3 doporučená a nejvýhodnější z hlediska integrace zařízení do větších celků (viz. kap. 3.3).

V této práci se především zaměřuji na část webového rozhraní, která splňuje **třídu 1** a **třídu 3**, tj. generovaná stránka je dynamická v HTML nebo textové podobě s podporou Webových služeb. Generování stránek v XML formátu nebude dále rozebíráno. Také otázka zabezpečení atributů proti neoprávněnému přístupu není řešena.

3.3 Topologie Web serveru

Je zde několik možných způsobů jak zaintegrovat webové rozhraní do PROFInet serveru, záleží na konkrétní aplikaci a možnostech hardwaru, na kterém je PROFInet server provozován.

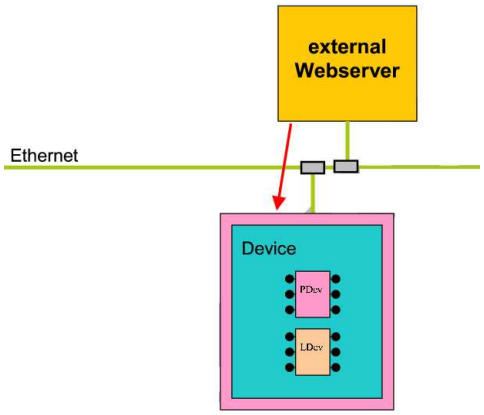
Integrovaný Webserver



Obrázek 3.1: Integrovaný web server

V této situaci je web server určen jen pro jedno konkrétní fyzické zařízení (obr. 3.1). Web server a fyzické zařízení mají stejnou IP adresu. Web server přistupuje k PROFInet komponentě přímo, přes rozšířené rozhraní PROFInetu. Vhodné řešení pro systémy, které nemají v operačním systému implementováno DCOM rozhraní.

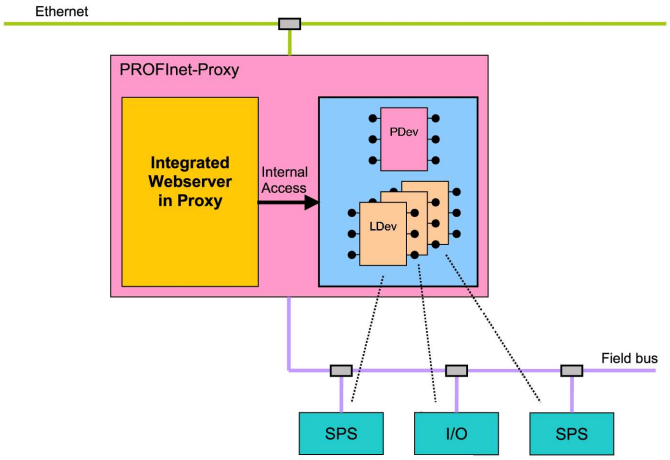
Externí Webserver



Obrázek 3.2: Externí web server

Web server není součástí PROFInet serveru, je spuštěn na jiném fyzickém zařízení, proto má i jinou IP adresu (obr. 3.2). Komunikace mezi nimi probíhá přes standardní rozhraní PROFInetu - DCOM. Web server může sdružovat informace z několika fyzických zařízení a vhodným způsobem je distribuovat.

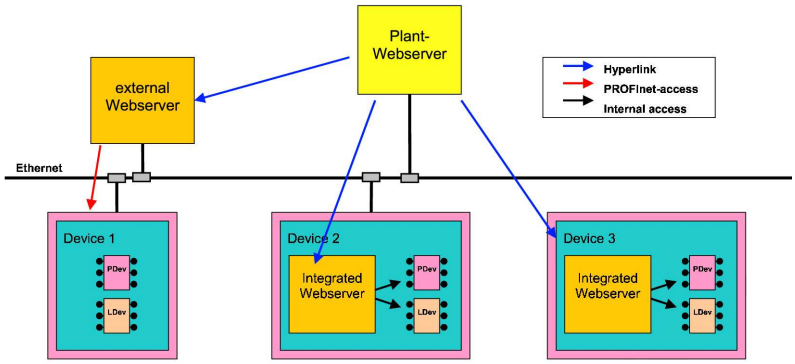
PROFInet proxy a Webserver



Obrázek 3.3: PROFInet proxy

Může být řešeno jako integrovaný nebo externí web server. Web server musí řešit přístup k několika fyzickým zařízením na jedné IP adrese, toto je řešeno přiřazením logického zařízení k reálnému fyzickému zařízení (obr. 3.3).

Plant Server



Obrázek 3.4: Konfigurace s plant serverem

Plant server shromažďuje informace z okolních PROFInet komponent přes jejich interní nebo externí web servery (obr. 3.4). Komunikace probíhá výhradně pomocí HTTP protokolu v kombinaci s HTML, případně XML. Plant server adresuje pouze web servery (jako "obyčejný" web prohlížeč) PROFInet zařízení. Web servery implementované na straně PROFInet zařízení musí být alespoň třídy 1 (tab. 3.1) a musí mít implementován mechanizmus pro dynamické získávání hodnot ze systému.

3.4 Adresní schéma

Adresování PROFInet objektu přes web rozhraní se sestává z adresy web serveru + adresy PROFInet objektu nebo atributu. Toto adresní schema je společné jak pro skriptovací rozhraní (kap. 3.6), tak i pro Webservices (kap. 4.5).

$$\boxed{\text{kompletní adresa} = \text{cesta k webserveru} + \text{adresa v PROFInet serveru}}$$

3.4.1 Syntaxe adresování v rámci PROFInet serveru

Adresování objektu v rámci PROFInet serveru:

$$\boxed{\textbf{PDev|LDev|RTAuto.Atribut.Connection}}$$

- **PDev, LDev, RTAuto** - jméno PDev, LDev a RTAuto zařízení, PDev může být nahrazeno IP adresou. Jednotlivé objekty jsou odděleny znakem "|".

- **Atribut** - jméno atributu PROFInet objektu. Objekt a atribut je oddělen znakem ”.”
- **Connection** - možný přístup k vlastnostem propojení[3] mezi objekty, název atributu obsahuje jméno proměnné objektu RTAuto. Atribut a Connection je odděleno znakem ”.”.

Extenzí *Connection* se můžeme dotázat na vlastnosti atributů objektu RTAuto v (tab. 3.2).

Connection =	**Možný výstup**	**význam**
ConnState	Up	Spojení s tímto atributem je vytvořené
	Down	Neexistuje spojení s tímto atributem
ConnChannel	DCOM	data se přenášejí přes DCOM
	Local	lokální propojení
	None	atribut není propojen
QC Quality Code	Good (NC)-OK	hodnota ok
	Good (C)-Local Override	
	Uncertain-Last usable value	použita poslední platná hodnota
	Uncertain-Substitute set	použita *substitue* hodnota
	Uncertain-QoS violation	alarmující QoS
	Bad	
	QC unknown-Value 0xabcd	

Tabulka 3.2: Vlastnosti propojení

Příklad adresování atributů RTAuto objektu

Následující příklad je ukázkou adresace atributu *CounterValue* objektu *RTAuto_Counter*, který je patří do LDev objektu *LDev_Counter*. Název PDev objektu je *suchac01*, ale lze použít i IP adresu zařízení.

suchac01|LDev_Counter|RTAuto_Counter.CounterValue

Další příklad rozšiřuje předcházející adresaci o dotaz na vlastnost *ConnChannel* atributu *CounterValue*.

suchac01|LDev_Counter|RTAuto_Counter.CounterValue.ConnChannel

[3] interconnections

3.4.2 Zápis do atributu RTAuto objektu

Pro přiřazení hodnoty do atributu je použito další klíčové slovo **value**, které je od názvu atributu odděleno znakem "&". Nová hodnota atributu je předána textové podobě.

PDev|LDev|RTAuto.Atribut**&value=hodnota**

Při úspěšném zápisu do proměnné vrátí webový server řetězec *"OK"* jinak je návratová hodnota typu *"103! Write error, internal error-code: 0xHRESULT"*, kde *HRESULT* je návratová hodnota funkce, která prováděla zápis.

Příklad zápisu do atributu

Provedeme zápis hodnoty *10* do atributu *Run*, RTAuto objektu *RTAuto_Counter*, patřící k logickému zařízení *LDev_Counter*, to vše zapouzdřené fyzickým zařízením na IP adrese *147.32.87.233*.

147.32.87.233|LDev_Counter|RTAuto_Counter.Run&value=10

3.4.3 Formát odpovědí z webového serveru

Formát dat, které web server vrací na naši žádost můžeme ovlivnit dalším klíčovým slovem **type**.

PDev|LDev|RTAuto.Atribut**&type=format**

Možné hodnoty klíčového slova **type** jsou v (tab. 3.3). V adresaci lze libovolně kombinovat klíčová slova **value** a **type**.

type=	**Popis**
plain	čistý text
html	text formátovaný do HTML stránky
xml	XML výstup podle specifikovaného standardu

Tabulka 3.3: Možný formát navrácených dat

3.5 Kompletní adresa

K adrese v rámci PROFInet serveru, popsané v (kap. 3.4.1), přidáme klíčové slovo **name**, oddělené od adresy web serveru znakem "?". V (tab. 3.4) je formální zápis.

http://IP_adr/profinet?name=PDev|LDev|RTAuto.Atribut&type=format

Tabulka 3.4: Formát kompletní adressy

Příklad kompletní adresy

Tento příklad ukazuje adresování atributu *CounterValue*, s požadovaným výstupem dat v *html* formě. Adresa (IP) web serveru nemusí být shodná s adresou fyzického zařízení PROFInet zařízení, záleží na použité topologii webového serveru (viz. kap. 3.3).

```
http://147.32.87.233/profinet?name=suchac01|LDev_Counter|
RTAuto_Counter.CounterValue&type=html
```

Další příklad požaduje zápis do atributu *Run*, s požadavkem o formátování potvrzení do *plain* formy. Webový server je spuštěn na vyhrazeným portu 8000.

```
http://147.32.87.233:8000/profinet?name=suchac01|LDev_Counter|
RTAuto_Counter.Run&value=1&type=plain
```

Tuto syntaxi lze již zapsat do adresního řádku webového prohlížeče.

3.5.1 Adresování více atributů současně

Lze použít této konstrukce pro adresování více atributů PROFInet serveru současně.

```
http://WebServeru?list=name="Device1" name="Device2"&type=format
```

Jednotlivé adresní sekvence začínající *name=* jsou od sebe odděleny znakem ' ' (mezera)[4]. Odpověď na tento soubor adresních sekvencí je zaslán ve stejném pořadí jako dotazy. Pod symbolickým názvem *DeviceX* se skrývá adresa v rámci PROFInet komponety popsané již dříve (kap. 3.4.1).

3.5.2 Adresování atributů PDev a LDev objektů

Jména atributů PDev a LDev objektů jsou pevně dána a jsou určena jen pro čtení. Oddělení PDev nebo LDev objektů od jména atributu je opět znakem ".", jako při adresování atributu RTAuto objektu. Podrobnější význam složitější atributů bude zmíněn později, ve vlastní implementaci webového rozhraní.

Name	jméno PDev objektu, je záměnné s IP adresou objektu
Producer	výrobce fyzického zařízení (hardwaru)
Product	jméno produktu
SerialNo	seriové číslo
Date	datum výroby nebo kompilace serveru
Revision	revize
LDevNames	seznam LDev objektů

Tabulka 3.5: Atributy PDev objektu

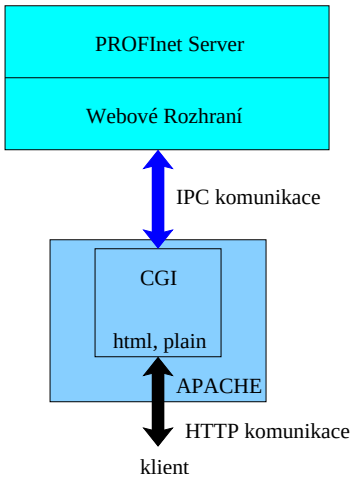
[4] lze použít i znak &

Name	jméno LDev objektu
Producer	producent softwaru
Product	jméno produktu
SerialNo	seriové číslo
Date	datum
Time	aktuální čas LDev objektu
Revision	revize
State	stav objektu (viz. kap. 1.2)
SetStateActive	přejdi do aktivního stavu
SetStateDeactive	přejdi do stavu připraveno
GroupError	informace funkčnosti objektu
ACCOInformation	informace o ACCO objektu, tzv. "ACCOPingFactor"
RTAutoNames	seznam RTAuto objektů, které vlastní tento LDev objekt
ConnectionNames	seznam názvů propojení mezi RTAuto objekty
ConnectionInfo	detailní informace o propojení

Tabulka 3.6: Atributy LDev objektu

3.6 Implementace webového rozhraní

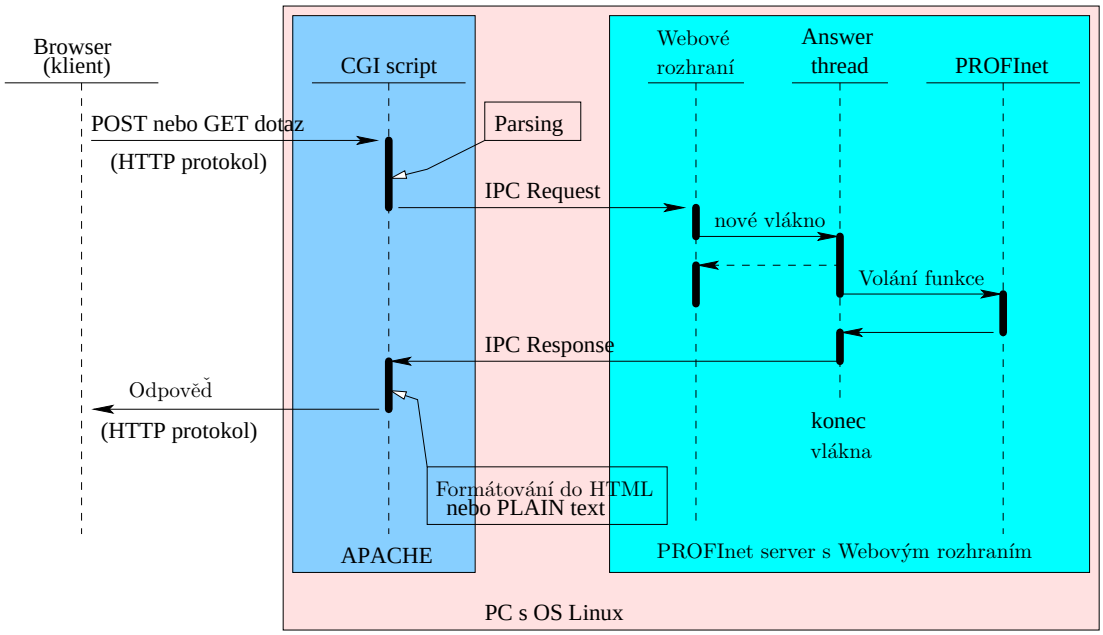
Byla zvolena implementace integrovaného web serveru (kap. 3.3), protože operační systém Linux nemá implementován DCOM. Ke stávající verzi PROFInetu je dopsáno rozšiřující rozhraní, které zprostředkovává přímou komunikaci mezi PROFInetem a vnějším CGI skriptem (obr. 3.5).



Obrázek 3.5: Blokové schema webového rozhraní

Obsluha klientských požadavků je znázorněna na (obr. 3.6), kde klient (web browser) předá dotaz CGI skriptu metodou POST, GET nebo jako obyčejný parametr dle specifikace

popsané výše (kap. 3.4). CGI skript provede "rozebrání" (parsing) tohoto řetězce a přes IPC message system posílá jednotlivé dotazy PROFInetu. Web rozhraní na straně PROFInetu vyvolá příslušnou metodu a její návratovou hodnotu pošle opět přes IPC zpět příslušnému CGI skriptu. CGI provede úpravu odpovědi do žádaného formátu (implementován HTML a "plain text" formát). Na straně PROFInetu je každý požadavek obsluhován zvláštním procesem (thread) s nastavenou nižší prioritou než má celý server.



Obrázek 3.6: Princip funkce webového rozhraní

3.6.1 Data od klienta

Čtení dat

Předání dat od klienta CGI skriptu může být uskutečněno několika způsoby. Nejednodušší způsob, kde se neúčasní web server(Apache), je přímo zavolání skriptu z příkazové řádky a předání dat v argumentu. Další možností je:

GET Při předávání metodou *GET* je web serverem nastavena proměnná prostředí *REQUEST_METHOD* na hodnotu **GET** a data jsou předány přes proměnou prostředí *QUERY_STRING*. Metoda je omezena ve velikosti předávaných dat.

POST Při metodě *POST* nastaví web server proměnou prostředí *REQUEST_METHOD* na hodnotu **POST**, dále je nastavena proměnná prostředí *CONTENT_LENGHT*, která udává velikost dat. Data čteme standardními funkcemi pro práci s datovými proudy[5] (*fread*, *fwrite*) ze standardního vstupu (stdin) [9]. Velikost předávaných dat není prakticky omezeno.

[5]stream

Metodu *POST* je vhodné využít při jednorázovém adresování více atributů PROFInet komponenty, nebo např. při zápisu textového řetězce do atributu RTAuto objektu, kdy velikost předávaných argumentů může být i několik kilobyte.

Zpracování dat

Po načtení dat od klienta do mezi paměti probíhá jejich rozebraní na jednotlivé adresní sekvence (syntaxe těchto sekvencí viz kap. 3.4). Přijatý datový blok je nejprve rozdělen na jednotlivé adresní sekvence a pak je každá sekvence rozdělena na konkretní části, které korespondují s názvy objektů a atributů v PROFInet serveru (adresní schéma PROFInet objektu viz. kap. 3.4.1).

Tyto informace jsou uloženy do datové struktury (obr. 3.8), která je následně poslána PROFInet serveru pomocí IPC komunikace.

3.6.2 IPC Komunikace

Názvem *IPC nástroje* nebo *System V IPC* se označují nástroje pro komunikaci mezi procesy. Tyto nástroje se prvně objevily v *Unixu AT&T System V.2* a obsahují funkce pro práci se semafory, sdílenou pamětí a komunikaci pomocí zpráv mezi procesy. Všechny tyto tři nástroje mají podobné funkční rozhraní. Základní funkce pro komunikaci jsou:

Ovládání *IPC message system* se děje pomocí těchto funkcí:

msgget - proces vytvoří frontu zpráv funkcí *msgget*. Vstupem této funkce je jedinečné číslo v rámci počítače a atributy, které nastavují vlastnosti fronty (např. práva přístupu). Pokud fronta v systému neexistuje je nutné nastavit atribut IPC_CREAT. Atributy se nastavují logickým součtem všech žádaných atributů. Návratovou hodnotou této funkce je *identifikátor fronty* zpráv.

msgsnd Funkce *msgsnd* umožňuje přidat do fronty novou zprávu. Funkci předáme *identifikátor fronty*, pointer na posílaná data a jejich velikost. Data, která chceme přenášet, musí mít formát podle obr. 3.7.

msgrcv Funkce *msgrcv* získá zprávu z fronty zpráv. Funkci předáme *identifikátor fonty*, ukazatel na alokovanou paměť, kam bude zpráva uložena a maximální velikost čtené zprávy. Posledním parametrem jsou atributy, kterými lze nastavit bude-li čtení z fronty blokující nebo ne (atribut IPC_NOWAIT). Při úspěšném provedení vrací funkce počet přijatých bytů.

msgctl Funkcí *msgctl* lze číst nastavení fronty zpráv, smazat zprávy čekající na vyzvednutí či frontu zpráv úplně odstranit ze systému.

Detailní popis IPC systému lze najít v [9]. Fronty zpráv se v použití podobají *pojmenovaným rourám*, odpadají však komplikace s otevíráním a zavíráním roury. Při používání zpráv se ovšem nevyhneme problémům, jako je blokování při plných frontách.

Maximální velikost zprávy je nastavena systémem na 8kb, maximální velikost fronty zpráv je nastavována PROFInet serverem pomocí makra MAXMSG_BUFFER.

```
struct zprava {
    long int typ_zpravy;

    /* Data, která chceme přenést */
};
```

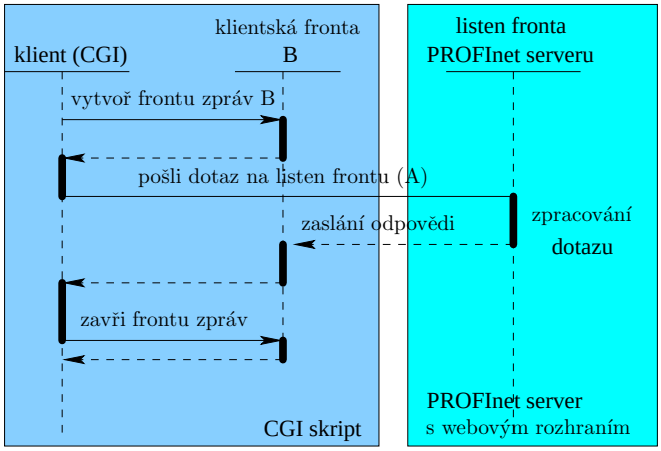
Obrázek 3.7: Struktura IPC zprávy

Komunikace mezi CGI a PROFInet serverem

Při komunikaci mezi CGI skriptem a webovým rozhraním na straně PROFInet serveru je právě použit *IPC message system*. Datová struktura, která je ve zprávě posílána je na obr. 3.8.

```
...struct... {
    integer ID;
    string  PDevName;
    string  LDevName;
    string  RTAutoName;
    string  Property;
    string  Value;
    string  Connection;
}...;
```

Obrázek 3.8: Struktura IPC zprávy použitá při komunikaci



Obrázek 3.9: Fronty Zpráv

Po rozebrání datového řetězce od klienta a uložení jeho jednotlivých částí do struktury (obr. 3.8) jsou vytvořeny dvě komunikační fronty zpráv. Fronta zpráv **A** (obr. 3.9) slouží

k napojení na *listen* frontu PROFInet serveru (vytvořena bez atributu IPC_CREAT, kap. 3.6.2) a poslání této datové struktury (request = požadavek), fronta zpráv **B** je vytvořena s *identifikátorem fronty* odpovídající PID[6] číslu klientského CGI skriptu[7]. PID číslo procesu je také součástí posílané datové struktury (obr. 3.8) v položce *ID*.

Po zpracování požadavku na straně PROFInet serveru (kap.3.6.3) se server připojí na klientskou frontu zpráv (fronta **B**)a zašle odpověď na požadavek. Odpověď na tento požadavek může být v některých případech větší, než je maximální povolená velikost zprávy. V takovém případě rozdělí server odpověď do 8kb paketů a ty pošle CGI skriptu postupně. Celkový počet a pořadí konkrétního paketu CGI skript zjistí z proměnné *typ_zpravy* obsažené ve struktuře odpovědi (obr. 3.7)

Formát odpovědi poslaný PROFInet serverem zpět CGI skriptu je řetězec znaků, kde jsou jednotlivé hodnoty odděleny znakem ";". CGI skript musí tato data interpretovat klientovi podle žádaného typu[8] výstupu (*plain text* nebo *html*)[9].

Formát výstupu "PLAIN TEXT"

Řetězec znaků od PROFInet serveru je přímo interpretován klientovi, jednotlivé žádosti (sekvence) jsou odděleny odřádkováním, jednotlivé hodnoty jsou odděleny znakem ";". Na začátek zprávy je CGI skriptem přidána informace o typu dat pro web server: *Content-Type: text/plain*.

Formát výstupu "HTML"

Jsou definovány implicitní stránky pro PROFInet objekty PDev, LDev, RTAuto a pro atributy těchto objektů (příloha 9 až 12). Řetězec s odpovědí od PROFInet serveru je rozebrán a vložen do těchto stránek. Vzhled je definován pomocí kaskádového stylu (CSS), větších změn lze dosáhnout změnou CGI skriptu. Na začátek zprávy je CGI skriptem přidána informace o typu dat pro web server: *Content-Type: text/html*.

Příklady formátování výstupu

```
http://147.32.87.233/profinet?list=
name=suchac01|LDev_Counter|RTAuto_Counter.Run
name=suchac01|LDev_Counter|RTAuto_Counter.CounterValue&type=plain
```

Jedná se žádost o hodnotu atributu *Run* a *CounterValue* v *plain* formě, obdržená odpověď bude tato:

```
Content-Type: text/plain

'1'
'17'
```

[6]Process Identification Number

[7]tím je zaručena jednoznačnost v rámci systému

[8]pokud není uživatelem specifikován typ výstupu, tak výstup je stejný jako by byl požadován "plain", ale neobsahuje hlavičku definující typ dat pro web server.

[9]formát *XML* není implementován

3.6.3 Struktura webového rozhraní

Server přijímá požadavky na listen IPC frontě, na kterou klient posílá datovou strukturu z obr. 3.8, kde *ID* je PID číslo klienta. Server po přijetí zprávy (obr. 3.9) vytvoří nový proces (*Answer_Thread*), kterému předá přijatá data a dál čeká na dalšího klienta. Tento nový proces vyhodnotí žádost a výsledek zašle zpět klientovi. Pak nastaví příznak, že jeho činnost byla ukončena, což pro rodičovský proces znamená, že může uvolnit jim používanou paměť a smazat ho ze seznamu aktivních procesů (seznam všech běžících *Answer_Thread*).

Obsluha klientské žádosti probíhá přes implementované funkční rozhraní, které se sestává z funkcí popsaných níže. Jsou rozděleny do skupin podle objektu, nad kterým pracují. Typy funkcí a argumentů korespondují s typy používanými v implementaci PROFInet serveru (dcom_s8 ∼ char, dcom_u8 ∼ unsigned char, atp.)

Physical device

Funkce pracující s objektem PDev mají jen výstupní argument(y).

- ```
  dcom_s32 GetPDevName(dcom_s8 *hostname, dcom_s8 *IPAddress);
  dcom_s8  GetPDevProducer(dcom_s8 *data);
  dcom_s8  GetPDevProduct(dcom_s8 *data);
  dcom_s8  GetPDevSerial(dcom_s8 *data);
  dcom_s32 GetPDevDate(dcom_s8 *data);
  dcom_s32 GetPDevRevision(dcom_s8 *data);
  ```
- `dcom_s32 GetPDevInfoList(dcom_s8 *str);`

  Funkce vrací všechny vlastnosti PDev zařízení v řetězci *data* odděleny znakem “;”.
- `dcom_s32 GetLDevList(dcom_s8 *str);`

  Funkce vrací seznam LDev objektů, které PDev obsahuje. Názvy jsou odděleny znakem “;”.

Logical device

Funkce pracující nad objektem LDev mají jako vstupní parametr datovou strukturu *profinet_items_t*, která je shodná se strukturou na obr. (3.8). Zde již je nutné adresovat konkrétní LDev zařízení, protože se jich na PDev objektu může nacházet několik.

- ```
  dcom_s8 *GetLDevName(profinet_items_t *pt);
  dcom_s8 *GetLDevProducer(profinet_items_t *pt);
  dcom_s8 *GetLDevProduct(profinet_items_t *pt);
  dcom_s8 *GetLDevSerial(profinet_items_t *pt);
  dcom_s8 *GetLDevRevision(profinet_items_t *pt);
  dcom_u16 GetLDevDate(dcom_u8 *str, profinet_items_t *pt);
  ```
- `dcom_s32 GetLDevInfoList(dcom_s8 *str, profinet_items_t *pt);`

  Funkce vrátí všechny vlastnosti PDev, popsané výše, v řetězci kde jsou jednotlivé hodnoty odděleny znakem “;”.

- `dcom_s32 GetPingFactor(dcom_s8 *value, profinet_items_t *pt);`

  Funkce vrátí hodnotu *ping faktor* ACCO objektu, jedná se o základní parametr ACCO objektu, viz [3].

- `STATEDEF GetLDevState(profinet_items_t *pt);`

  Funkce vrací stav LDev, může nabývat těchto hodnot: *NonExistent, Initializing, Operating, Ready, Defect.*

- `dcom_s32 GetLDevTime(dcom_s8 *Value,profinet_items_t *pt);`

  Informace o aktuálním čase LDev objektu, může být nastaven zcela nezávisle na PDev objektu a i na ostatních LDev objektech.

- `GROUPERRORDEF GetLDevGroupError(profinet_items_t *pt);`

  Funkce vrátí informaci o funkčnosti LDev: *NonAccessiable, Ok, Problem, Unknown.*

- `dcom_s32 GetConnectionInfo(dcom_s8 *buff, conpair_list_t *ptList, profinet_items_t *pt);`

  Detailní informace o propojení, v řetězci vrátí položky oddělené znakem ";": ID propojení; provider; jméno atributu providera; jméno propojení; datový typ propojení; hodnota; stav propojení; komunikační kanál; kvalita (QoS).

- `dcom_s32 GetConnectionNames(dcom_s8 *buff, conpair_list_t *ptList, profinet_items_t *pt);`

  Funkce vrátí seznam názvů navázaných spojení s okolními objekty.

- `dcom_s32 SetLDevState_Deactivate(profinet_items_t *pt);`

  Uvede LDev objekt (a tím i jeho RTAuto objekty) do stavu *Ready*, tj. objekt zpracovává vstupy, ale výstupy jsou zmraženy.

- `dcom_s32 SetLDevState_Activate(profinet_items_t *pt);`

  Uvede LDev objekt (a tím i jeho RTAuto objekty) do stavu *Operating*, tj. normální pracovní stav objektu.

- `dcom_s32 GetRTAutoList(dcom_s8 *list, profinet_items_t *pt);`

  Seznam jmen RTAuto objektů.

RunTime Automation Object

- `HRESULT GetPropertyValue(dcom_s8 *value, dcom_u8 *ucQC, profinet_items_t *pt);`

  Vrací hodnotu atributu RTAuto objektu.

- `HRESULT PutPropertyValue(dcom_s8 *data, dcom_u8 ucQC, profinet_items_t *pt);`

  Umožňuje modifikaci atributu RTAuto objektu.

- `dcom_s32 GetPropertiesNames(dcom_s8 *buff, list_t *ptList, profinet_items_t *pt);`

  Seznam všech atributů RTAuto objektu v řetězci sestaveném následovně: jméno proměnné; její datový typ; práva modifikace;.

Tyto funkce přímo volají funkce nebo přistupují k datovým strukturám PROFInetu. Pro představu doporučuji nahlédnout do zdrojových kódu webového rozhraní.

3.6.4 Způsob obsluhy žádosti

V okamžiku vytvoření procesu *Answer_Thread* přebírá tento proces kontrolu nad zpracováním požadavku. Procesu je předána přijatá struktura, která specifikuje žádost. Podle počtu vyplněných políček nejprve *Answer_Thread* určí, nad kterým objektem se bude pracovat, výběr provede pomocí konstrukce *switch(...)* a *case* (obr. 3.10). Z objektů PDev a LDev je možné atributy jen číst, podle jména žádaného atributu zavolá proces příslušnou funkci (kap. 3.6.3).

Atributy nad objekty RTAuto je možno jak číst tak do nich zapisovat, jako indikace zápisu musí být proměnná *Value* (obr. 3.8) nenulová.

```
ptRequest = ...;  //ukazatel na přijatou strukturu s požadavkem
 switch(typZadosti) {

  case PDev:     // dotaz nad PDev objektem
            ...
            if(!strcmp(atribut,''Producer''))
                GetPDevName(buffer);
            ...
  break;

  case LDev:    //dotaz nad LDev objektem
            ...
            if(!strcmp(atribut,''Product''))
                buffer=GetPDevName(ptRequest);
            ....
  break;

  case RTAuto: //dotaz nad RTAuto objektem
             ...
  break;
}
```

Obrázek 3.10: Obsluha žádosti webovým rozhraním

Soubory s implementací webového rozhraní jsou pevnou součástí PROFInet serveru. Při jakékoliv změně v kódu web rozhraní je nutno překompilovat celý server.

Soubory s implementací webového rozhraní		
./v1.2-src/dcomap/test/ linux/webint	*IPCmsg.c*	IPC komunikace, Answer_Thread()
	webint.c	funkce pro komunikaci s PROFInetem
	webint.h	deklarace funkcí
	web_cfg.h	společné datové struktury CGI skriptu a webového rozhraní

Zapnutí webového rozhraní v PROFInet serveru je uskutečněno spuštěním samostatného procesu, které vytvoří listen frontu a čeká na žádosti od CGI skriptu.

3.6.5 Struktura CGI skriptu

CGI skript převezme požadavek (argumenty) od klienta, vytvoří napojení na listen frontu zpráv na PROFInet server, zašle požadavek a čeká na odpověď (popsáno v kap. 3.6.2). V případě adresování více atributů současně (kap. 3.5.1), je vytvořen dynamický seznam s jednotlivými dotazy, který je postupně zasílán na webové rozhraní PROFInet serveru. Schéma zpracování požadavku je zobrazeno na obr. (3.11).

Funkce *SendRequest()*[10] a *WaitForResponse()* mají nastaveny časové limity jak dlouho se funkce opakuje při neúspěšném odeslání/přijetí[11].

Funkce pro formátování výstupu do *html* nebo *plain* textové podoby přihlížejí na typ dotazu (např. dotaz na hodnotu atributu RTAuto objektu, PDev objektu atp.)

Soubory s implementací CGI skriptu		
./cgi	*profinet-cgi.c*	vlastní CGI skript
	format.c	funkce pro formátování výstupu do HTML nebo PLAIN, defaultní stránky objektů PDev, LDev, RTAuto.
	parse.c	funkce pro rozebrání požadavku od CGI skriptu
	web_cgi.h	deklarace funkcí CGI skriptu

3.7 Postup sestavení aplikace, konfigurace

Po z kompilování CGI skriptu *gcc* [8] kompilátorem jej nakopírujeme do adresáře, ve kterém může Apache server spouštět CGI skripty (obvykle adresář /cgi-bin). Cesta k tomuto adresáři musí být nastavená v makru *CGI_BIN* v konfiguračním souboru ./dcamap/test/linux/webint/web_cfg.h, dále musí být ve stejném souboru nastaveno makro *CSS_NAME*, které se odkazuje na použitý kaskádový styl (CSS). Detailnější konfigurací Apache serveru bude ukázána při konfigurování aplikace modelu akvária.

[10] ve skutečnosti se jedná o několik funkcí, zde je tato množina nazvána souhrnně takto, jen pro ilustraci problému

[11] režim příjímání je v tzv. neblokujícím režimu [9]

```
// provedeme rozebrání adresního řetězce na jednotlivé
// a uložíme do dynamické struktury, která odpovídá struktuře
// pro komunikaci mezi CGI a webovým rozhraním profinetu

  Cutting(pBegin, request);

// pointer pBegin ukazuje na začátek dynamického seznamu
// datových struktur

pList = pBegin;
while (pList->next != NULL) { //dokud není seznam prázdný

   SendRequest(pList); // pošle prvek seznamu na webové rozhraní
                       // PROFInetu pomocí IPC message systému

   WaitForResponse(pResponse); // čeká na odpověď s nastaveným
                               // časem pro vypršení žádosti

   FormatToPLAINOutput(pResponse, Output);
// nebo
   FormatToHTMLOutput(pResponse, Output);

   printf(``%s``, Output);

pList=pList->Next;
}
```

Obrázek 3.11: Struktura CGI skriptu pro webové rozhraní

3.8 Shrnutí

IPC komunikace byla zvolena pro svojí flexibilitu a rychlost. Komunikace není závislá na představeném CGI skriptu, ten může být nahrazen jiným, napsaným ve skriptovacím jazyce, např. v Perlu nebo PHP. Jediné, co je nutné zachovat, je datová struktura pro výměnu dat. Webové rozhraní, které je součástí PROFInet serveru, bude třeba časem upravovat a doplňovat o další funkce, postupně s rozvojem PROFInetu jako celku. Toto je hlavní nevýhoda oproti použití externího webového serveru s použitím komunikačního protokolu DCOM.

CGI skript je spouštěn Apachem, webovým serverem, jeho konfigurace a veškeré náležitosti pro spuštění celé aplikace budou ukázány na modelu akvária v závěru této práce.

Kapitola 4

Webové služby

4.1 Úvod

Web Services se do českého jazyka překládá jako *Webové služby*, ale tento název není zcela ekvivalentní, neboť se nejedná jen o služby poskytované přes web. Web Services je následník systému vzdáleného volání procedur v distribuovaných systémech jako jsou RPC, CORBA[1], RMI[2]. Oproti svým předchůdcům není tak vyzrálý (neřeší prozatím například autentizaci), ale jedná se o jednoduchý, otevřený a zcela platformě nezávislý komunikační systém. Technologii Webových služeb tvoří tři části:

- komunikační protokol pro vzdálené volání procedur *SOAP*[3], který přenáší data zapsaná v XML jazyku.

- jazyk pro popis poskytovaných služeb, zvaný *WSDL*[4]

- mechanismus pro nalezení služeb[5], používají se dva standardy *UDDI* a *WSII*

4.2 SOAP

Protokol SOAP vznikl v roce 1998 ve firmě Microsoft, později se na jeho rozšíření podílelo mnoho firem, včetně IBM. Paralelně s ním vzniklo mnoho podobných protokolů jako XML-RPC, WDDX, XMI. Verze SOAP 1.2 byl přijat konsorciem W3C.

Protokol SOAP posílá argumenty volané funkce metodou *POST* a jsou zapsány pomocí XML jazyka. Pomocí XML jazyka si lze předávat jak jednoduchý text, tak i složité objekty a struktury. Při použití *XML schéma* lze navíc jednotným a rozšiřitelným způsobem definovat strukturu a typ předávaných dat. Pomocí *XML NameSpace* lze zabránit kolizím mezi stejnými názvy pro různé typy.

---

[1] **C**ommon **O**bject **R**equest **B**roker **A**rchitecture
[2] **R**emote **M**ethod **I**nvocation
[3] **S**imple **O**bject **A**ccess **P**rotocol
[4] **W**eb **S**ervice **D**escription **L**anguage
[5] v této práci se jimy nebudeme zabívat

SOAP zpráva se skládá ze třech částí: *Envelope*, *Header* a *Body*. *Envelop* (obálka) je kořenovým tagem celé zprávy. V něm je obsažena část uvozená tagem *Header* (hlavička zprávy), tato část je nepoviná, pomocí ní lze nastavit parametry komunikace. Za hlavičkou zprávy následuje povinná část uvozená tagem *Body* (tělo zprávy), které již obsahuje jméno volané metody a její argumenty.

Původní implementace přenášela SOAP zprávy protokolem HTTP, v současnosti existují implementace i pro přenos pomocí protokolu SMTP[6].

4.2.1 Příklad SOAP komunikace

Mějme například funkci *boolean jePrvocislo(long cislo);*, která má číselný argument *cislo* a vrací pravdivostní hodnotu podle toho, zda *cislo* je či není prvočíslo. Vzhledem k tomu, že webová služba muže být implementována v libovolném jazyce, jsou typy *boolean* a *long* definovány v XML schema a ne nutně přítomné v konkrétním jazyce. SOAP vyžaduje, aby každá funkce byla v určitém jmenném prostoru. U objektově orientovaného jazyku je tímto prostorem objekt a funkce odpovídá jeho metodě. U jazyků, které nejsou objektově orientované nemá jmenný prostor přímou interpretaci, ale musíme nějaký definovat. Jmenný prostor je určen URI adresou. Příklad SOAP zprávy je na (obr. 4.1).

```
<env:Envelope
  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
  env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xs="http://www.w3.org/1999/XMLSchema"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
 <env:Body>
  <m:jePrvocislo xmlns:m="urn:mojeURI" >
    <cislo xsi:type="xs:long">1987</cislo>
  </m:jePrvocislo>
 </env:Body>
</env:Envelope>
```

Obrázek 4.1: SOAP zpráva pro volání funkce

Zpráva je přenesena na server, který funkci implementuje pomocí protokolu HTTP a metodou POST (nejčastěji). Funkce na straně serveru je jednoznačně určena svým jménem a jmenným prostorem. Ve volání metody na (obr. 4.1) jsou celkem definovány čtyři prefixy jmenných prostorů: *env* použitý pro tagy samotného SOAPu, *xs* pro XML schema, *xsi* pro instanci XML schema a námi definovaný *m* jako jmenný prostor naší funkce. Názvy prefixů lze volit libovolně, důležité je, ke kterým URI se váží.

[6]Simple Mail Transport Protocol = protokol určený na doručování elektronické pošty

```
<env:Envelope
  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<env:Body>
 <ns1:jePrvocisloResponse
   xmlns:ns1="urn:mojeURI"
   env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <return xsi:type="xsd:boolean">true</return>
 </ns1:jePrvocisloResponse>
</env:Body>
</env:Envelope>
```

Obrázek 4.2: SOAP zpráva pro vrácení návratové hodnoty

4.3 WSDL

Jazyk WSDL (Web Services Description Language) je určen pro definici nabízených služeb vzdáleným serverem. Jazyk je velice obecný, aby byla zachována platformní nezávislost. Nejednodušší bude ukázat popis jednoduché služby na příkladu (obr. 4.3).

WSDL dokument obsahuje popis jedné služby (klíčové slovo *service*), což je největší jednotka. Jedna služba má jednu, nebo několik bran (klíčové slovo *port*). Každá brána má vazbu (klíčové slovo *binding*), což je způsob, jak se daná brána volá (např. SOAP přes HTTP) a nějakou přístupovou adresu(URL). Lze tedy mít pro jednu službu více bran s různými vazbami, tj. volat jednu službu různými způsoby, např. první brána: SOAP přes HTTP, druhá brána: SOAP přes HTTP nad SSL atp. Vazby odkazují na rozhraní (*portType*), které je souhrnem operací (*operation*). Rozhraní v objektově orientovaném jazyku odpovídá objektu, jednotlivé operace odpovídají metodám nad tímto objektem.

Každá operace definuje obvykle dvě zprávy (*message*), jednu vstupní (*input*) a jednu výstupní (*output*). Každá zpráva může obsahovat několik částí (*part*), které odpovídají argumentům a návratovým hodnotám.

Dokument v jazyce WSDL příslušející k Webové službě ji plně popisuje, pokud tedy máme WSDL popis služby, můžeme službu plně využívat (obvykle ke stažení ze serveru, který službu poskytuje).

4.4 Nástroje pro implementaci

Z freewarových nástrojů je asi nejznámější *Apache Axis* pracující v Javě, který zahrnuje WSDL generátor a také nástroj pro generování kostry funkcí z WSDL souboru.

Za nejrychlejší (nejrychleji reagující na požadavky) je považován nástroj *gSOAP* [7], který pracuje s jazykem C/C++. Zatímco Axis dokáže SOAP zprávy generovat dynamicky v okamžiku, kdy konkrétní službu potřebuje, gSOAP provádí generování SOAP zpráv v průběhu kompilace zdrojových kódů bez možnosti změny za běhu programu. Je to kompro-

```
<message name="jePrvocisloRequest">
 <part name="cislo" type="xsd:long" />
</message>

<portType name="Cisla">
 <operation name="jePrvocislo" parameterOrder="cislo">
  <input  message="m:jePrvocisloRequest" name="jePrvocisloRequest"/>
  <output message="m:jePrvocisloResponse"  name="jePrvocisloResponse"/>
 </operation>
</portType>

<binding name="cislaSoapBinding" type="m:Cisla"> ... </binding>

<service name="CislaService">
 <port binding="m:cislaSoapBinding" name="cisla">
  <wsdlsoap:address location="http://nekde.cz/cisla"/>
 </port>
</service>
```

Obrázek 4.3: Popis služby pomocí jazyka WSDL

mis mezi flexibilitou a extrémní rychlostí celé aplikace. Obsahuje taktéž generátor WSDL popisu služby a inverzní generátor kostry programu z WSDL souboru (zatím ve fázi vývoje, nepodporuje všechny konstrukce jazyka WSDL).

Pro úplnost lze jmenovat i komerční nástroje jako je *WebSphere (IBM)*, *Oracle9iAS (Oracle)*, *.NET (Microsoft)*.

4.5 Implementace Webových služeb

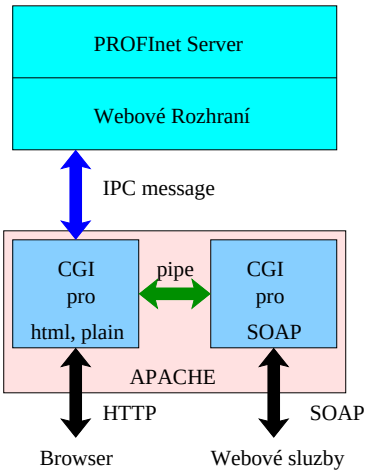
K implementaci webových služeb byl využit projekt gSOAP [7], který přímo pracuje v jazyku C a je volně šiřitelný. Vytvoření aplikace pomocí této implementace je snadné a rychlé.

Aplikace pro webové služby je také tvořena CGI skriptem, který obsluhuje SOAP komunikaci s klientem. Pro přístup k datům z PROFInetu využívá již dříve popsaný CGI skript (kap. 3.6.5), který spouští s přesměrovaným standardním výstupem do *roury*[7] [9], ze které čte data z PROFInetu a ty poskytuje klientovi pomocí protokolu SOAP (obr. 4.4).

Hlavní součástí gSOAPu je překladač *soapcpp2*, který z hlavičkového souboru nabízených funkcí (nazván *profinet.h*) vygeneruje kostru webové služby, *xml* strukturu jednotlivých dotazů (request) a odpovědí (response). Součástí generovaného kódu je i WSDL popis služby. Nejdůležitější z generovaných souborů jsou tyto:

- **soapClient.c** - připravené kostry jednotlivých služeb pro stranu klienta (proxy),

[7] tzv. pipe



Obrázek 4.4: Struktura rozhraní pro webové služby

každá ze služeb nejprve provede *serializaci* (popsáno dále) argumentů metody, pak odeslání na server, který službu poskytuje. Po přijetí odpovědi provede *deserializaci* odpovědi. Tento soubor se přidává jen ke klientské aplikaci.

- **soapServer.c** - připravené kostry webových služeb pro stranu serveru (stub), po *deserializaci* přijaté zprávy do konkretních argumentů je zavolána požadovaná funkce, výsledek je *serializován* a zaslán nazpět klientovi. Přidává se k aplikaci, která služby poskytuje.

- **soapC.c** - funkce pro *serializaci* a *deserializaci* společné jak straně klienta tak serveru.

Názvem *serializace* je nazván proces, který zkonvertuje argumenty funkce do XML struktury. Proces *deserializace* je postup obrácený. Informace lze najít v dokumentaci k projektu gSOAP [7].

4.5.1 Nabízené služby

Seznam nabízených služeb definujeme v hlavičkovém souboru, označen *profinet.h* jeho součástí je také definice jmenného prostoru. Protože jazyk C není objektový, je příslušnost proměnné či metody k určitému jmennému prostoru vyjádřena definovaným operátorem "__" (dvě podtržítka). Na obr. 4.5 je ukázka deklarace nabízených služeb pro přístup k PROFInetu. Prvních několik řádků není komentář, překladač *soapcpp2* je zpracovává a získává z nich informace o jmenných prostorech, jméno generovaného WSDL souboru, kde se bude skript nalézat (důležité pro WSDL soubor), atd.

Dále máme definovanou strukturu *LDevNames_t* (obr. 4.5), která patří do jmenného prostoru *ns* a obsahuje ukazatel na datový typ *LDev_Name*. Struktura má speciální tvar (viz. [7]) pro definici dynamických polí (předem nevíme kolik objektů LDev PDev objekt obsahuje), velikost pole je určena *__size*. Název datového typu položky pole je vhodné si

```
//gsoap ns service name: profinet
//gsoap ns service namespace: http://suchac01/profinet.wsdl
//gsoap ns service location: http://suchac01/cgi-bin/
//gsoap ns service executable: profinet-soap.cgi
//gsoap ns schema namespace: urn:profinet

typedef char* ns__LDevName;

typedef struct ns__LDevNames_tag {
 ns__LDevName *__ptr;
 int __size;
} ns__LDevNames_t;

int ns__GetLDevNames(char *PDevName, ns__LDevNames_t **LDevNames);
...
```

Obrázek 4.5: ukázka ze souboru *profinet.h* pro webové služby

nadefinovat, neboť se promítne ve vygenerovaném XML dotazu či odpovědi.

Specifikaci nabízených služeb pro PROFInet je možno nalézt v dokumentaci [4].

Nabízené služby pro objekt PDev

Následující funkce mají vstupní argument jméno PDev objektu, výstupním argumentem je datová struktura. Pro více informací nahlédněte do výše zmíněného hlavičkového souboru s nabízenými službami. Adresování jednotlivých objektů v rámci PROFInet komponenty je shodné s webovým rozhraním (kap. 3.4.1).

- **int ns__GetPDevInformation()** - načte kompletní informace o objektu PDev do datové struktury.
- **int ns__GetLDevNames()** - získává seznam všech LDev objektů do dynamického pole řetězců.

Nabízené služby pro objekt LDev

Vstupním argumentem následujících služeb je jméno LDev objektu.

- **int ns__GetLDevInformation()** - vrací datovou strukturu s kompletními informacemi o LDev objektu.
- **int ns__GetGroupError()** - informace o funkčnosti objektu.
- **int ns__GetACCOInformation()** - Informace o ACCO objektu.

- **int ns__GetState()** - stav objektu.
- **int ns__SetState()** -nastav stav objektu (Operate nebo Ready).
- **int ns__GetRTAutoNames()** - jména podřazených RTAuto objektů.

Nabízené služby pro RTAuto objekt

- **int ns__GetRTAutoPropertyNames()** - vstupním argumentem je jméno RTAuto[8] objektu, výstupem je dynamická struktura s atributy objektu.
- **int ns__GetProperty()** - vstupem je atribut RTAuto objektu, výstupem hodnota atributu
- **int ns__SetProperty()** - vstupem je atribut RTAuto objektu a jeho nová hodnota hodnota, výstupem je informace zda-li byl zápis úspěšný.

Nabízené služby pro informace o propojení

- **int ns__GetConnectionQuality()** - vstupem je jméno propojení, vrací jeho QoS (Quality of Servis).
- **int ns__GetConnectionState()** - vstupem je jméno propojení, vrací jeho stav.
- **int ns__GetConnections()** - vrací jména všech propojení nad konkrétním LDev objektem.
- **int ns__GetConnectionsInfo()** - detailní informace o propojení mezi objekty nad konkrétním LDev objektem.

Datové položky struktur, které jsou návratové hodnoty popsaných funkcí lze nalézt v souboru *profinet.h*.

4.5.2 generování SOAP zpráv

Na obr. 4.6 je ukázán vygenerovaný SOAP dotaz pro službu zapsanou na obr. 4.5. Tag *<ns:GetLDevNames>* určuje jméno volané služby. Mezi tagy *<PDevName>* a *</PDevName>* je vloženo jméno PDev objektu. V záhlaví dotazu je načtení jmených prostorů pro SOAP, XML a náš jmenný prostor *ns* - profinet.

Na dotaz (obr. 4.6) obdržíme odpověď na obr. (4.7). Dotaz můžeme zaslat na port webového serveru demonstračně například pomocí *telnet* klienta.

Při použití gSOAP knihovny se tvorbou zpráv na úrovni SOAP protokolu nemusíme zabývat, vše probíhá zcela transparentně. Uživatel, který chce využívat některou ze vzdálených služeb, potřebuje znát jen WSDL soubor, který obsahuje její popis, z něj pomocí aplikace *wsdlcpp*[9] vygeneruje hlavičkový soubor, který již muže kompilovat *soapcpp2* překladačem. Program *wsdlcpp* je ve vývoji a nezvládá některé složitější konstrukce.

[8] jménem RTAuto objektu je myšlena kompletní cesta v rámci komponety, tedy včetně PDev a LDev objektu

[9] součást gSOAP projektu

```
POST /cgi-bin/profinet-soap.cgi HTTP/1.0
Host: 192.168.0.100
Content-Type: text/xml; charset=utf-8
Content-Length: 529
SOAPAction: ""

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:ns="urn:profinet">
 <SOAP-ENV:Body id="_0" SOAP-ENV:encodingStyle="
             http://schemas.xmlsoap.org/soap/encoding/">
  <ns:GetLDevNames>
   <PDevName>192.168.0.100</PDevName>
  </ns:GetLDevNames>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Obrázek 4.6: Kompletní dotaz v SOAP protokolu, včetně HTTP hlavičky

4.5.3 Typ aplikace

Aplikace vytvořená pro obsluhu webových služeb může být vytvořena dvěma způsoby:

- **CGI skript**, který je volán nadřazeným webovým serverem jako klasický CGI skript.
- **Stand Alone server**, který je zcela autonomní. Získáme kompaktní, výkonou aplikaci pro práci se SOAP protokolem. Interpretaci webových stránek, podporu cgi skriptů, posílaní dat (připadá v úvahu jen metoda GET), je nutno do implementovat.

Postup implementace je podobný, liší se v algoritmu, který vyvolává jednotlivé služby (s menšími úpravami lze přejít od jednoho typu k druhému). Jak již bylo napsáno v úvodu, pro účely PROFInetu byl zvolen první z jmenovaných způsobů, tedy webové služby pomocí CGI skriptu.

4.5.4 Struktura CGI skriptu

Funkce *main()* CGI skriptu pro obsluhu webových služeb obsahuje pouze tyto funkce:

```
int main() {
  struct soap soap;
  soap_init(&soap);
```

```
HTTP/1.1 200 OK
Date: Fri, 26 Dec 2003 20:33:58 GMT
Server: Apache/1.3.28 (Unix)
Connection: close
Content-Length: 678
Content-Type: text/xml; charset=utf-8

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="
           http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns="urn:profinet">
<SOAP-ENV:Body SOAP-ENV:encodingStyle="
           http://schemas.xmlsoap.org/soap/encoding/"
id="_0">
<ns:GetLDevNamesResponse>
  <LDevNames xsi:type="ns:LDevNames-t">
     <item xsi:type="ns:LDevName">LDev_Counter</item>
     <item xsi:type="ns:LDevName">LDev_Calculator</item>
     <item xsi:type="ns:LDevName">LDev_DataTypeMirror</item>
  </LDevNames></ns:GetLDevNamesResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Obrázek 4.7: Kompletní odpověď v SOAP protokolu, včetně HTTP hlavičky

```
    soap_serve(&soap);
    soap_end(&soap);
    return(0);
  }
```

Funkcí *soap_init()* provedeme inicializaci proměnných prostředí (struktura *soap*) pro vlastní SOAP komunikaci na implicitní hodnoty. Mezi proměnnými je např. nastavení verze HTTP protokolu, časové limity pro příjem, port pro stand alone server, komprese zpráv, SSL podpora atd. Jejich případnou změnu můžeme provést přímým zápisem do datové struktury *soap*.

Funkce *soap_serve()* přijme požadavek od klienta, provede *deserializaci* (získání argumentů metody ze SOAP zprávy) zprávy a zavolá příslušnou službu (metodu). Výstup metody je *serializován* do SOAP zprávy a zaslán zpět klientovi. Tato funkce je součástí generovaného souboru *soapServer.c*.

Funkce *soap_end()* zavře používaný soket v *soap* struktuře, uvolní paměť.
Funkce pro inicializaci (*soap_init()*) a konec (*soap_end()*) SOAP komunikace jsou obsaženy v knihovně projektu gSOAP, v souboru *stdsoap2.c*.

Dále by měla následovat implementace jednotlivých služeb deklarovaných v hlavičkovém souboru (*profinet.h*). Pro naší ukázkovou funkci *GetLDevNames()* (obr. 4.5) máme tuto vygenerovanou kostru:

```
int  ns__GetLDevNames(struct soap     *soap,
                      char            *PDevName,
                      ns__LDevNames_t **LDevNames) { ... }
```

V těle této funkce zavoláme spuštění CGI skriptu webového rozhraní (kap. 3.6.5) s přesměrovaným výstupem do roury. CGI skriptu předáme parametry jako při volání příkazu z příkazové řádky, nelze použít metodu POST - standardní vstup, protože ten používá samotný SOAP protokol (před voláním skriptu je nutné deaktivovat proměnou prostředí REQUEST_METHOD a po provedení příkazu ji opět nastavit na hodnotu POST (kap. 3.6.1)). Výsledek načtený z roury uložíme do datové struktury *LDevNames*. Pro úspěšné dokončení služby nesmíme zapomenout na volání *return(SOAP_OK)*.

Více se o implementaci služby nemusíme starat, příjem dat, volání služby a zaslání odpovědi zaopatří metoda *soap_serve()*.

Soubory s implementací webových služeb		
./cgi-soap	*profinet-soap.c*	CGI skript pro webové služby
	pn_client-aqa.c	ukázková implementace klienta
	profinet.h	deklarace nabízených služeb

4.5.5 Volání služby z klientské aplikace

Následuje ukázka zdrojového kódu klientské aplikace, která využívá službu *GetLDevNames()*. Název metody volané klientem má o proti metodě definované na straně serveru

prefix *soap_call_*. Jedná se jen o rozšíření původní metody o další argumenty. Částečná definice této metody je v obsažena v souboru *soapClient.c*. Klient má informace pouze o argumentech metody a použitých datových typech, o těle metody neví nic.

```
int main() {
  struct soap soap;
  ns__LDevNames_t *LDevNames;
  int i;

 soap_init(&soap);
 soap_call_ns__GetLDevNames(
     &soap,
     "http://192.168.0.100:8000/cgi-bin/profinet-soap.cgi",
     "",
     "192.168.0.100",// jméno PDev
     &LDevNames      // návratová hodnota
     );
   if(soap.error) soap_print_fault(&soap,stderr);
    else for(i=0;i<LDevNames->__size;i++)
  printf("%s\n",LDevNames->__ptr[i]);

soap_end(&soap);
 }
```

Službu poskytuje server na adrese uvedené ve druhém argumentu volané funkce. Výstupem bude pole datových struktur *LDevNames_t* (definice na obr. 4.5) o délce *__size*.

4.6 Shrnutí

Vytváření webových služeb pomocí projektu gSOAP je velice rychlý, před programátorem je naprosto skryt přenos zpráv mezi poskytovatelem a klientem, (de)serializace (přijatých)odesílaných zpráv (z)do XML jazyka, navazování spojení atp. Pro uživatele není rozdíl mezi voláním lokální funkce a použití funkce distribuované v rámci webových služeb.

Webové služby pro PROFInet poskytují tomuto standardu další komunikační rozhraní pro získávání informací o technologii na platformách, kde není možné použít DCOM protokol. Volání funkcí pomocí webových služeb lze lépe zakomponovat do aplikací psaných např. v jazyku C nebo Java než při použití webového rozhraní, které je spíše určeno pro vizualizaci technologie.

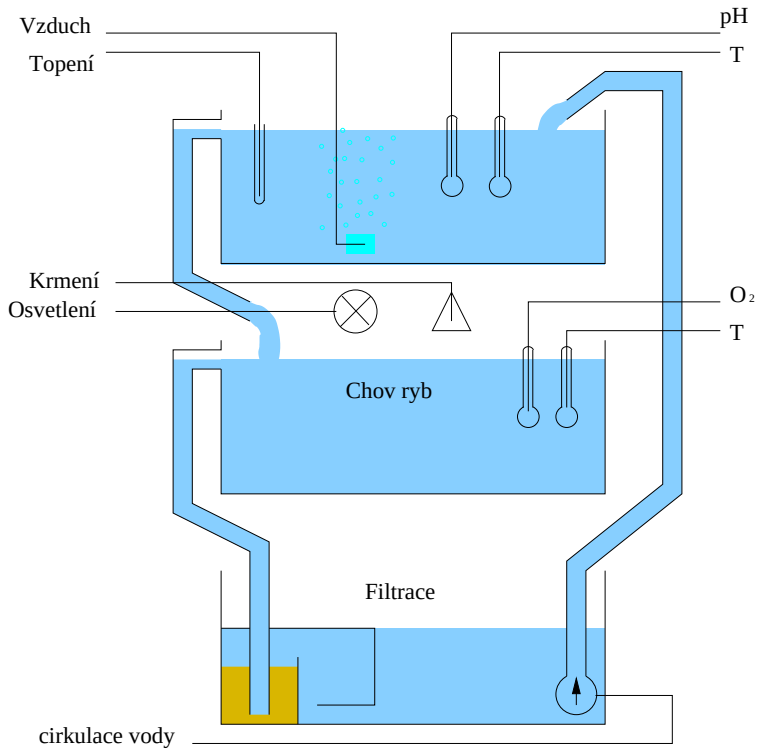
Implementovanými funkcemi nebyly vyčerpány všechny vlastnosti PROFInetu, které by bylo vhodné distribuovat či pomocí webových služeb ovlivňovat. Jedná se spíše o jeden z prvních pokusů o implementaci, sloužící k demonstračním účelům.

Kapitola 5

Model akvaria

V této kapitole se seznámíme s modelem akvária na kterém bude ukázáno využití webového rozhraní a webových služeb. Podrobný popis tvorby PROFInet komponenty pro OS Windows byl popsán v [1], postup je totožný s Linuxem.

5.1 Popis modelu



Obrázek 5.1: Schema akvária

Model akvária je tvořen třemi nádobami kaskádně propojenými (obr. 5.1), ze třetí nádoby je voda přečerpávána zpět do první nádoby. První nádoba slouží jako přípravna

vody, druhá je pro chov ryb a třetí je pouze filtrační. Instalované snímače a akční členy na modelu jsou tyto:

První nádoba - přípravna vody[1]**:**

- měření teploty (T1).

- měření pH.

* třístupňové topení.

* vzduchování.

Druhá nádoba - chov ryb

- měření teploty (T2).

- měření okysličení vody.

* krmička.

* osvětlení akvária.

Třetí nádoba - filtrace

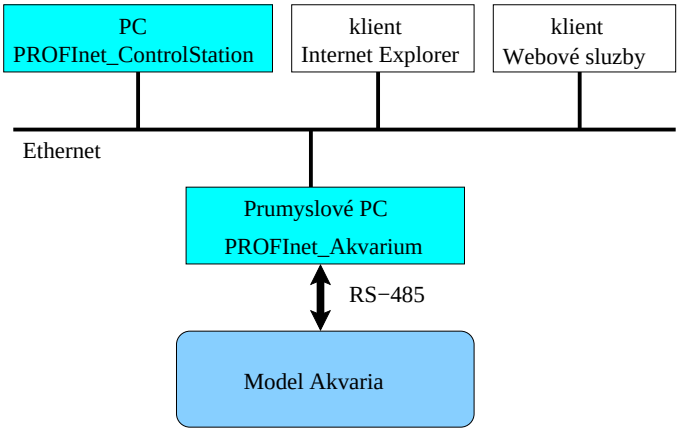
* čerpadlo pro cirkulaci vody.

Snímače a akční členy komunikují s průmyslovým počítačem prostřednictvím sběrnice RS-485 (obr. 5.2). Na průmyslovém počítači je spuštěna PROFInet komponenta *Akvárium*, která komunikuje pomocí ovladače (popsáno dále) se snímači a akčními členy umístěnými na modelu, jedná se o transformační uzel (proxy) mezi RS-485 a PROFInetem. Na tomto počítači neprobíhá generování akční veličiny, to má na starosti komponenta *Control*, která na základě změřených hodnot ovládá vytápění a okysličování akvárií, z této stanice lze také měnit žádané veličiny a spouštět jednorázové krmení. Jedná se o pomalou soustavu.

5.1.1 Přechod od RS-485 k PROFinetu

Na průmyslovém počítači na kterém běží PROFInet server je v pozadí spuštěn program - ovladač, který zajišťuje přesun hodnot ze snímačů do sdílené paměti a zpět, ze sdílené paměti do akčních členů. Snímače a akční členy jsou připojeny ke sběrnici RS-485. Jsou cyklicky čteny všechny snímače na modelu a jejich hodnoty ukládány do sdílené paměti, ke které mohou přistupovat další procesy. Tento ovladač je součástí diplomové práce [12]. Spuštění ovladače je automatické při spuštění komponenty Akvária[2].
Sdílená paměťová oblast pro výměnu dat má podobu:

[1] akční členy označeny "*", snímače "-"
[2] využit příkaz *fork()*



Obrázek 5.2: Síťová topologie

```
typedef struct share_memory4exchange_tag {
  //obraz senzoru
    int basin1_temperature;
    int basin1_pH;
    int basin2_temperature;
    int basin2_oxygen;

  //obraz akcnich clenu
    unsigned short pump;
    unsigned short light;
    unsigned short heating_unit;
    unsigned short oxygenator;
    unsigned short feeding;
 } share_memory4exchange_t;
```

Ovladač sběrnice RS485 provadí zápis do proměných obrazu senzorů, PROFInet aplikace zapisuje do obrazu akčních členů. Proces, který provádí zápis či čtení z této oblasti, nastaví semafor (vstup do kritické sekce) aby byla zachována konzistence dat. Ovladač provádí cyklické čtení senzorů, zápis do akčních členů provádí jen tehdy, pokud obdrží od PROFInet aplikace signál.

Signálem pro zápis do akčních členů byl opět použit sdílený semafor, který je nastaven komponentou Akvárium tehdy, jeli požadován zápis do akčních členů.

Hodnoty ze snímačů jsou čteny v celočíselné podobě, desetiná čárka má pevné místo: u hodnoty teploty dělíme 10, u hodnoty pH a okysličení dělíme 100. Tuto úpravu provádí PROFInet komponenta akvária a dále tyto hodnoty distribuuje v podobě desetiného čísla (float).

Funkce pro práci se sdílenou pamětí

Funkce pro práci se sdílenou pamětí opět spadají do IPC nástrojů představených již dříve. Sdílená paměť je speciální skupina adres, které vytváří IPC pro jeden proces a objeví se v adresovém prostoru tohoto procesu. Jiné procesy si potom mohou tento segment sdílené paměti připojit ke svému vlastnímu adresovému prostoru.

- **shmget()** - podobně jako u IPC signálů vrátí identifikátor sdílené paměti, jenž je pak využíván ostatními funkcemi.
- **shmat()** - připojení sdílené paměti k aktuálnímu procesu.
- **shmdt()** - odpojí sdílenou paměť od aktuálního procesu (paměť ale zůstane zachována, není rušena).
- **shmctl()** - funkce pro řízení sdílené paměti (i její rušení).

Použití semaforů

Pro práci se semafory se použijí tři funkce:

- **semget()** - inicializuje sdílený semafor a vrací na něj identifikátor. Současně nastaven atribut (SEM_UNDO), aby při ukončení procesu, operační systém zaručil, uvolnění semaforu. To znamená, pokud je proces při náhlém ukončení v kritické sekci, operační systém tuto sekci odemkne.
- **semop()** - operace se semaforem, tato funkce umožňuje změnu jeho hodnoty o definovanou velikost (obvykle +1 nebo -1). Pokud by hodnota semaforu po provedení této operace měla mít zápornou hodnotu, je vstup do kritické sekce zakazán - provádění procesu je pozastaveno (BLOKING_MODE), nebo funkce jen vrátí chybový kód a ošetření nechá na programátorovi (NONBLOKING_MODE).
- **semctl()** - pomocí této funkce mužeme nastavit semafor na konkretní hodnotu, nebo semafor odstranit z paměti.

5.2 Návrh PROFInet komponent

Budeme se zabývat programovým řešením PROFInet komponent použitých k řízení modelu a vytvoření reprezentace komponenty do návrhového prostředí iMap. Vzájemné propojení mezi RTAuto objekty je naznačeno v příloze 13. Specifikace činnosti PROFInet komponenty popsaná dále se definuje v souboru *sysobj.c*.

5.2.1 Vytvoření IDL popisu komponenty

Vytvořený IDL soubor s popisem rozhraní komponenty je nejprve kompilován překladačem *midl* (součást Micro$oft Visial Studia):

midl /Oicf idl.idl.

Výstupem je kromě jiných soubor *idl.tlb*, který je vstupem do konvertoru *AMCvtTlb* (součást PROFInetu):

AMCvtTlb /noprototypes /prefix=SR idl.tlb.

Parametr */noprototypes* zakazuje generování prototypy funkcí pro jejich přímé volání, parametr */prefix=SR* určuje prefix u názvů generovaných datových struktur, který musí korespondovat s označením v PROFInet komponentě.

Po této konverzi získáme soubory *idl_t.h* a *idl_t.c*, ve kterých jsou datové struktury s definicí rozhraní RTAuto objektů, viz [2]. Tento soubor je staticky zahrnut do kódu komponenty (popsáno v kap. 2.9).

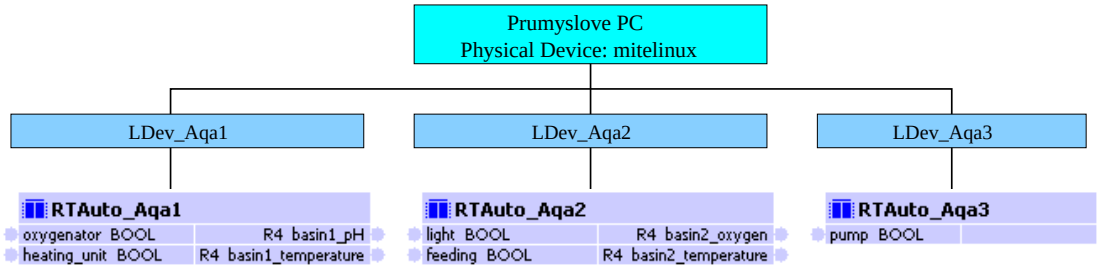
5.2.2 Vytvoření reprezentace komponenty pro iMap

Pro vytvoření XML souboru, který je využit v iMapu jako reprezentace komponenty byl použit program *Component Editor*[3]. Program má "přívětivé" uživatelské rozhraní, po zadání názvu komponenty, přímo definujeme názvy vstupních/výstupních atributů RTAuto objektů a jejich datové typy. Výstupem tohoto programu je XML soubor, který lze vložit do knihovny komponent v iMap prostředí. Strukturu xml souboru lze nalézt v [4].

Značnou nevýhodou tohoto programu je, že nedochází k současnému generování IDL souboru. Programátor tak musí dvakrát definovat to samé rozhraní, vždy v jiném jazyku.

5.2.3 Komponenta Akvárium

Každé z akvárií je reprezentováno jedním RTAuto objektem. Z důvodu omezení PROFInet serveru v1.2 na počet RTAuto objektů které mohou být v jednom LDev objektu (poměr 1:1), připadá na každé akvárium jeden LDev objekt (obr. 5.3). V PROFInet komponentě je



Obrázek 5.3: Komponenta akvárium

zaregistrován uživatelský časovač, který periodicky spouští *Callback* (kap. 2.8) funkci, a ta čte hodnoty snímačů ze sdílené paměti a ukládá je do výstupních atributů RTAuto objektů. Naopak vstupní atributy RTAuto objektů zapisuje do sdílené paměti, kde je přebírá a dále distribuuje ovladač až po zaslání signálu (viz předešlá kapitola). Tento signál je vázán na funkci *SysLinux_Property_Put()*[4], která je v PROFInet komponentě vyvolána jen tehdy,

[3] ke stažení na stránkách www.profibus.com
[4] viz soubor *sysobj.c*

pokud se mění vstupní atributy RTAuto objektu. Struktura řídící smyčky je znázorněna na obr. (5.4).

```
void DCOM_FKT_HUGE SysLinux_ScanCycleCallBack(...) {
  SysLinux_ReadValueFromShareMemory(&getvalue);
  ...
  // zapsat hodnoty z načtené struktury ‘‘getvalue’’
  // do výstupních atributů RTAuto objektu
  ...
}
```

Obrázek 5.4: Řídicí smyčka pro komponentu *Akvárium*

Význam atributů komponenty Akvárium

V tab. 5.1 jsou atributy RTAuto objektů komponenty Akvárium.

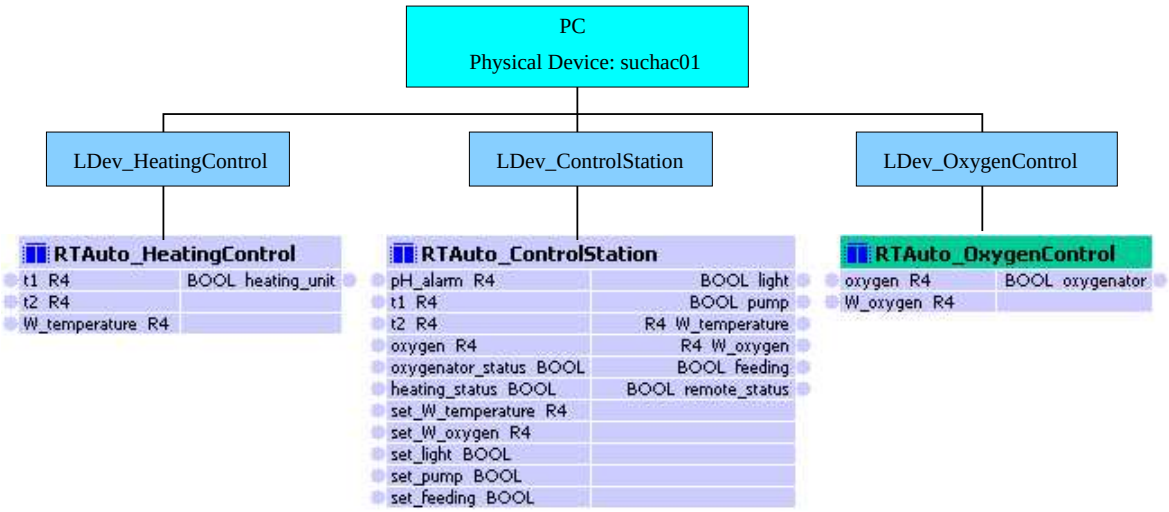
Atribut	**Datový typ**	**I/O**	**Význam**
RTAuto_Aqa1			
oxygenator	bool	in	ovládání vzduchování
heating_unit	bool	in	ovládání vytápění
basin1_pH	float	out	hodnota pH
basin1_temperature	float	out	teplota vody
RTAuto_Aqa2			
light	bool	in	osvětlení
feeding	bool	in	krmička
basin2_oxygen	float	out	obsah kyslíku ve vodě
basin2_temperature	float	out	teplota vody
RTAuto_Aqa3			
pump	bool	in	ovládání čerpadla

Tabulka 5.1: Atributy komponenty Akvárium

5.2.4 Komponenta Control

Komponenta je rozdělena do tří RTAuto objektů (obr. 5.5):

- **HeatingControl** - řízení vytápěcí jednotky.
- **OxygenControl** - řízení okysličení vody.
- **ControlStation** - komponenta využitá k vizualizaci a k nastavení žádaných hodnot - teploty a okysličení vody.



Obrázek 5.5: Komponenta Control

Výstup objektů (akční veličiny) *HeatingControl* a *OxygenControl* se generuje jen tehdy, pokud se mění některý z jejich vstupů, proto je výhodné akce těchto RTAuto objektů zahrnout do funkce *SysLinux_Property_Put()*, která je systémem PROFInetu volána, pokud některý provider chce provést zápis do atributu RTAuto objektu. Vstupní atributy objektu *ControlStation* jsou zapsány do "lokálních" proměnných (obrazy vstupů).

Pro objekt *ControlStation* je do PROFInet systému zaregistrován časovač s *Callback* funkcí, která cyklicky přepisuje informace (v textové podobě) na monitoru a generuje hodnoty výstupních atributů.

Význam atributů komponenty Control

V tab. 5.2 jsou atributy RTAuto objektů komponenty Control. RTAuto objekt *RTAuto_HeatingControl* má informaci o teplotách v první nádrži (t1) a v nádrži 2 (t2) a hodnotu žádané teploty v druhém akváriu (W_temperature). Podle velikosti odchylky t1 a t2 od žádané hodnoty je ovládáno vytápění první nádrže.

Objekt *RTAuto_OxygenControl* vyhodnocuje odchylku žádané hodnoty okysličení od skutečné hodnoty a podle ní ovládá (zapnuto/vypnuto) vzduchování první nádrže.

Objekt *RTAuto_ControlStation* je určena k vizualizaci a k nastavování žádaných hodnot. Atributy s prefixem *set_* jsou určeny k nastavení žádaných hodnot z externí komponenty nebo např. přes webové rozhraní. Tyto atributy jsou akceptovány jen tehdy, pokud je komponenta přepnutá do stavu *vzdáleného ovládání*. Jinak je možné žádané hodnoty ovlivnit jen z textového rozhraní (terminálu) této komponenty. Stav ovládání (remote/local) je signalizováno atributem *remote_status* a je možné ho nastavit jen z textového rozhraní komponenty Control.

Atributy s postfixem *_status* odpovídají stavu akčních členů, využito pro jejich monitorování. Ovládání krmení (feeding), osvětlení (light) a cirkulace vody (pump) je pouze manuální, není řízeno automaticky.

Atribut	Datový typ	I/O	Význam
RTAuto_HeatingControl			
t1	float	in	teplota v nádrži 1
t2	float	in	teplota v nádrži 2
W_temperature	float	in	žádaná hodnota teploty
heating_unit	bool	out	ovládání vytápění
RTAuto_OxygenControl			
oxygen	float	in	stav okysličení
W_oxygen	float	in	žádaná hodnota okysličení
oxygenator	bool	out	ovládání oxygenatoru
RTAuto_ControlStation			
pH_alarm	float	in	hodnota pH
t1	float	in	teplota v nádrži 1
t2	float	in	teplota v nádrži 1
oxygen	float	in	stav okysličení vody
oxygenator_status	bool	in	oxygenator zapnut/vypnut
heating_status	bool	in	vytápění
set_W_temperature	float	in	žádaná hodnota teploty
set_W_oxygen	float	in	žádaná hodnota okysličení
set_light	bool	in	osvětlení zapnuto/vypnuto
set_pump	bool	in	cirkulace vody
set_feeding	bool	in	krmička
light	bool	out	akční člen
pump	bool	out	akční člen
feeding	bool	out	akční člen
W_temperature	float	out	akční člen
W_oxygen	float	out	akční člen
remote_status	bool	out	status vzdáleného ovládání

Tabulka 5.2: Atributy komponenty Control

5.3 Spuštění komponenty

Pro úspěšné spuštění komponenty je nutné provést tuto sekvenci příkazů:

- **BASE_Startup()** - inicializace trasovacího systému, inicializace kritických sekcí (v některých systémech potřeba).
- **RpcInit()** - inicializace RPC systému, spuštění časovače, který odpojuje klienty kteří neodpovídají do časového limitu.
- **SysLinux_DComOsInit()** - inicializace DCOM systému, nastavení UUID čísla (kap. 1.3) celé komponenty.
- **SysLinux_DComOsOpen()** - otevření portu, spuštění *ping* threadu (kap. 2.5).
- **SysLinux_Device_Startup()**
- **SysLinux_Startup()** - vytvoří objektový model PROFInetu - PDev s jedním implicitním LDev objektem, které je později nahrazen LDev objektem definovaný uživatelem.
- **SysLinux_InitUserObjects()** - vytvoří objekty LDev, první LDev objekt nahradí implicitně definovaný předchozí funkcí. Dále inicializuje uživatelské RTAuto objekty i s jejich metodami.
- **SysLinux_StartTimerThread()** - spustí proces, který obsluhuje uživatelské časovače.
- **SysLinux_AttacheToShareMemory()** - u komponenty akvárium se připojíme ke sdílené paměti.
- **SysLinux_StartScanCycle()** - zaregistrujeme uživatelský časovač pro cyklus čtení/zápis proměnných komponenty.
- **SysLinux_StartPersist_Thread()** - zvláštní proces, který ukládá informace o propojení mezi RTAuto objekty do pevné paměti.
- **SysLinux_Persist_InitConnections()** - pokud se v pevné paměti nachází informace o propojení mezi objekty, dojde k jejich načtení a opětovnému zavedení.
- **SysLinux_All_LDev_OnStateChanged()** - do této chvíle byli všechny LDev objekty ve stavu *Ready*, výstupy byly na definovaných hodnotách (implicitně nulové), tímto příkazem přejdeme do stavu *Operate*, normální režim LDev objektu.
- **msgCGI_Init()** - spuštění hlavního procesu pro webové rozhraní.

Po těchto inicializačních příkazech je vytvořeno několik kooperativně pracujících procesů. Ukončovací sekvence příkazů je analogická k inicializačním funkcím (opačném pořadí než při spouštění). Sekvence spouštění a ukončení běhu komponenty je zahrnuta v souboru *main.c*.

5.4 Konfigurace

Pro interpretaci webových stránek byl zvolen webový server Apache verze 1.3.28, s těmito zakompilovanými moduly: **http_core** - neodělitelná část webového serveru; **mod_cgi** -podpora CGI skriptů; **mod_include** - podporu SSI[5]; **mod_alias** - pro možnost použití aliasů v konfiguračním souboru serveru; **mod_access** - řízení přístupu.

Server je spuštěn na portu 8000 (zvoleno). CGI skripty jsou směřovány do adresáře ./cgi-bin.

5.4.1 Komponenta Akvárium

Tato komponenta je spuštěna na průmyslovém počítači. V adresáři */home/profinet* se nacházejí všechny náležitosti nutné pro běh PROFInet komponenty spolu s webovým rozhraním a službami. Struktura adresářů je znázorněna v tab. 5.3. Pro komunikaci se

Adresář	Soubor	Popis
./htdocs	profinet.shtml	výchozí stránka PDev objektu
	query.html	dialog pro ruční zadání dotazu
	profinet.css	definice kaskádového stylu
./cgi-bin	**profinet.cgi**	skript pro webové rozhraní
	profinet-soap.cgi	skript pro webové služby
./	**PROFInet_Aqarium**	PROFInet komponenta
	httpd	Apache server
	httpd.conf	konfigurační soubor Apache

Tabulka 5.3: Adresářová struktura komponenty Akvárium

snímači a akčními členy se současně s komponentou spouští ovladač sběrnice RS-485, název ovladače je *aquacomm* a předpokládá se, že je uložen v adresáři */home/rs485*.

PROFInet komponentu je nutné spustit s právy uživatele root, webový server může být spuštěn pod libovolným uživatelem, automaticky přejde pod uživatele *apache*.

Průmyslový počítač, na kterém běží tato komponenta má k dispozici 16Mb paměti RAM a 16Mb paměti FLASH (užitá jako FLASH disk). Proto je komponenta sestavena s nejnižší možnou trasovací úrovní (výsledný kód je o půlku menší než při plném trasování) a Apache server jen s nejnutnějšími moduly, které jsou zahrnuty ve spustitelné části Apache serveru. Logovací zprávy o běhu Apache server jsou nastaveny na minimální hodnotu a směřovány do RAM disku[6], protože FLASH disk je při běhu systému v režimu *pouze pro čtení* (RO), do režimu *zápis povolen* (RW) se přejde speciálním příkazem.

5.4.2 Komponenta Control

Komponenta Control je spuštěna na "normálním" počítači s operačním systémem Linux (použit RedHat v8.0) s jádrem 2.4.18. Komponenta muže být po svém přeložení spuštěna

[5] **S**erver **S**ide **I**nclude

[6] oblast v paměti RAM chovající se jako pevný disk

přímo z adresáře *./gen* (popsáno dále), není nutné žádné další nastavování.

5.4.3 Sestavení aplikace

Obě komponenty byli sestaveny překladačem *gcc* verze 2.95.3, některé části pro komponentu akvária museli být kompilovány *staticky*[7], kvůly problémům s rozdílnou knihovnou *glibc* na obouch systémech. Na obr. 5.4 je adresářová struktura, výchozí adresář je */$(HOME)/PROFInet*[8].

Adresář	**Popis**
./Aqaria.Pn	zdrojové soubory komponenty Akvarium
./Aqaria.iMap	komponenty pro iMap
./Aqaria_Control	zdrojové soubory komponenty Control
./RS485	zdrojové soubory ovladače RS485
./gen	generované spustitelné soubory
./lib	knihovy PROFInetu
./v1.2-src	zdrojové soubory PROFInetu, přeložené *.o soubory jsou ukládány do adrseře ./lib
./cgi	zdrojové soubory CGI skriptu pro webové rozhraní
./cgi-soap	zdrojové soubory CGI skriptu pro poskytování webových služeb
./apache	webový server Apache

Tabulka 5.4: Adresářová struktura pro sestavení komponent

Součástí zdrojových souborů PROFInetu je i webové rozhraní (adr. *./dcomap/test/linux/webint*), které jsou po přeložení nakopírovány s ostatními *.o soubory do adresáře *./lib*. Každý z částí (adresářů) má svůj vlastní *Makefile* soubor pro přeložení zdrojových kódů.

5.5 HTML stránky modelu

Pro vizualizaci modelu akvária lze díky webovému rozhraní použít i klasický webový prohlížeč. Díky SSI podpoře ve webovém serveru Apache, lze kdekoliv v HTML kódu stránky použít konstrukce:

```
<!- -# include virtual="/cgi-bin/profinet.cgi?name=...&type=html" - ->
```

[7]knihovny, které jsou jinak k aplikaci "přidány" až při spuštění se staticky přidají ke spustitelnému kódu aplikace. Důsledkem je mírný nárůst velikosti aplikace.

[8]důležité při sestavování aplikace, nutné zachovat

Pokud má Apache server zapnutou podporu SSI, při interpretaci stránky uživateli nechápe tento výraz jako komentář[9], ale provede specifikovanou akci (zavolání cgi skriptu s parametry) a výsledek tohoto skriptu vloží místo tohoto komentáře. Vzhled takovéto stránky je v příloze 14. Popis všech klíčových slov (příkazů), použitelných v SSI výrazech lze nalézt [13].

5.6 Užití webových služeb

Byl napsán jednoduchý klient využívaný k získání základních informací o modelu webových služeb. Program je možné sestavit pod operačním systémem Linux a Microsoft Windows, je napsán v jazyce C a využívá knihovny z projektu gSOAP [7]. Provádí vizualizaci - jednorázově načte kompletní informace o PROFinet komponentě. Klient demonstruje platformní nezávislosti webových služeb. Verze pro operační systém Windows je sestavena v prostředí Microsoft Visual Studio.

5.7 Shrnutí

Na modelu akvária jsem demonstroval základní vlastnosti při použití webového rozhraní a webových služeb. Protože se jedná o pomalou aplikaci, lze rozdělit generování akční veličiny a přímí kontakt s technologií do dvou komponent propojených ethernetem (TCP/IP).

Stabilita komponent byla testována v konfiguraci uvedené v příloze 13. Dále byl ze dvou stanic sledován stav webovými prohlížeči, průběžně byli získávány souhrné informace pomocí webových služeb za současného monitorování pomocí prostředí iMap.

Mezi další podpůrné programy pro PROFInet od společnosti PROFIBUS patří *PROFInet Test Tool*, který je spuštěn pod operačním systémem Windows 2000 a který umožňuje zasílat dotazy na všechny definované rozhraní PROFInetu.

[9] v jazyce HTML je komentář uvozen symboly: <!- - můj komentář - ->

Závěr

Tato práce přináší souhrn poznatků při adaptování PROFInet serveru na operační systém Linux, implementaci webového rozhraní pro přístup k PROFInet objektům pomocí HTTP protokolu a rozhraní pro zpracování webových služeb při komunikaci protokolem SOAP. K demonstrování těchto vlastností byly vytvořeny dvě PROFInet komponenty, které kooperativně řídí reálný model akvária.

Adaptace PROFInetu do systému Linux byla před mým dokončením uveřejněna na stránkách organizace PROFIBUS verzí adaptace firmy IFAK (Německo). Výsledný kód samotného PROFInetu použit v této práci je z části převzat od firmy IFAK. Získané znalosti při adaptaci mi napomohly k lepšímu pochopení jednotlivých částí PROFInet serveru. Linuxová implementace je plně funkční a lze s ní nahradit implementaci pod operačním systémem Windows. Byli testovány základní vlastnosti chování při výpadku komunikace mezi dvěmi objekty (použití definované nebo poslední platné hodnoty). Komunikace s návrhovým prostředím od firmy Siemens, iMap, je funkční v plném rozsahu, tj. konfigurace a následné nahrání propojení mezi objekty, konfigurace QoS, monitorování *on-line*.

Přístup pomocí HTTP protokolu je možný běžným internetovým prohlížečem. Aktuální informace z PROFInetu jsou získávány přes webové rozhraní, které je tvořeno CGI skriptem, který komunikuje přímo s PROFInetem. Jako webový server byl použit Apache v minimální konfiguraci (jen s podporou CGI a SSI). Pro vizulizaci akvária byla vytvořena webová stránka, která je cyklicky obnovována a která zobrazuje informace o modelu i o objektech PROFInetu - informace o propojení, diagnostika zařízení apod. Díky SSI podpoře v Apache serveru lze tuto dynamickou stránku vytvořit velice snadno. Popsané webové rozhraní splňuje všechny specifika popsané v [4].

Komunikace s komponentou pomocí SOAP protokolu je vhodná pro programy vytvořené např. v jazyku C nebo Java, které jsou zpracovávány na platformě, na které nelze použít standardního komunikačního protokolu PROFInetu, DCOM. Byly implementovány služby, které pokrývají distribuci základních vlastností PROFInetu. Velikou výhodou SOAP protokolu je, že přenášená zpráva je zapsána ve srozumitelné formě jak pro počítač, tak pro člověka, v jazyce XML. Klient webových služeb může být napsán libovolným nástrojem, který umí z WSDL popisu služeb vygenerovat příslušné funkce.

Dalším krokem při rozšiřování webového rozhraní a služeb by mělo být:

- zabezpečení proti neoprávněnému přístupu a to zejména přístup k atributům RTAuto objektů, které v této verzi může kdokoliv přes web modifikovat.

- implementovat webové služby za pomoci gSOAP knihovny jako *stand alone* server by ušetřilo další, jinak využitelné paměťové místo v průmyslovém počitači - nemusel by se použít webový server Apache pro vyvolání skriptu.

□

Literatura

[1] Kamil Vyštejn: *Diplomová práce - PROFInet*, ČVUT FEL, Praha 2003.

[2] *PROFInet - Implementation Guide v1.2*, July 2002, URL: http://www.profibus.com.

[3] *PROFInet - Architecture Description and Specification v1.0*, August 2001, URL: http://www.profibus.com.

[4] *PROFInet - Architecture Description and Specification v2.0* (kapitola 4), January 2003, URL: http://www.profibus.com.

[5] Dalibor Kačmář: *Programujem v COM a COM+*, Computer Press 2000

[6] Inline Assembler URL: http://www.manualy.sk/skolycky/skolicka26.txt

[7] Elektronická dokumentace k projektu *gSOAP 2.3*, URL: http://www.cs.fsu.edu/∼engelen/soap.html.

[8] Projekt *gcc*, URL: http://www.gnu.org

[9] Richard Stones, Neil Matthew: *Linux, začínáme programovat*, Computer Press 2000.

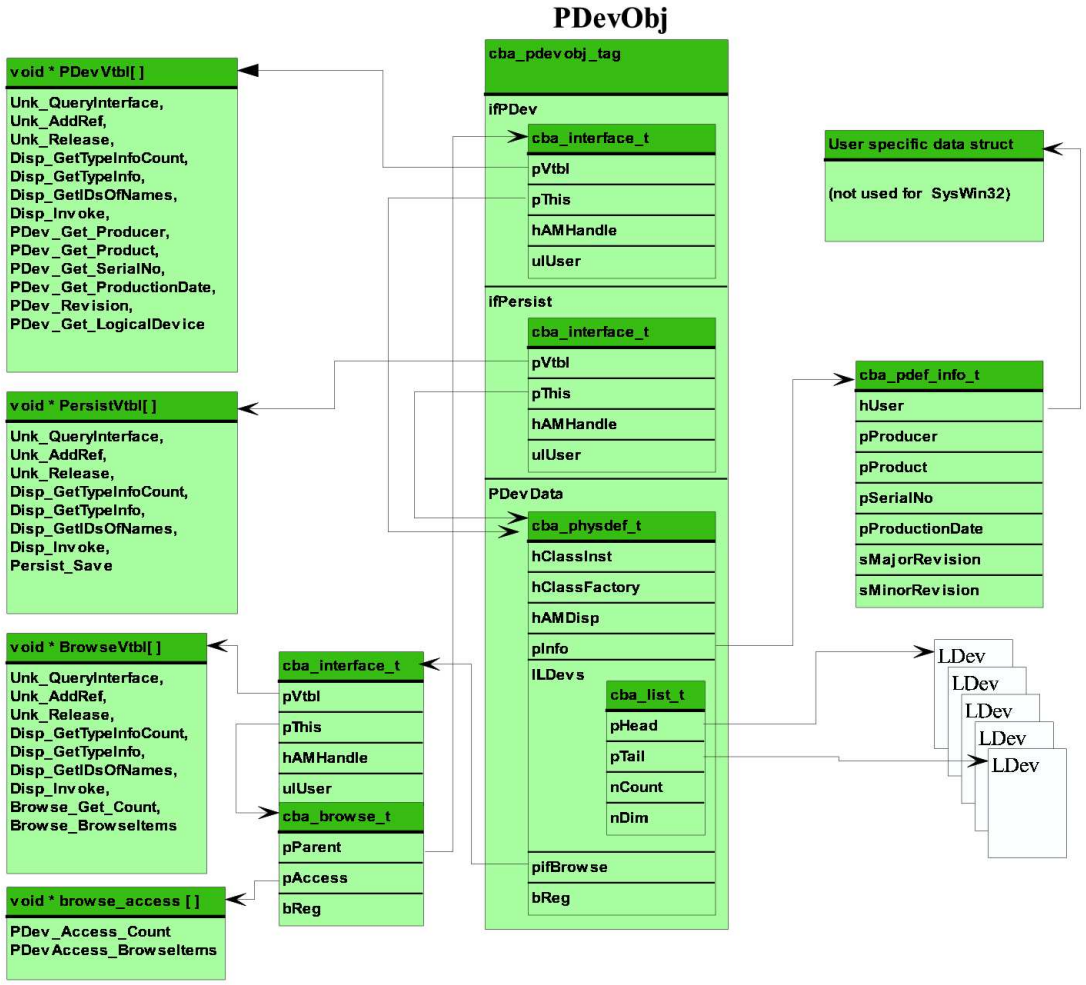
[10] Bill O. Gallmeister: *POSIX.4: Programing for the Real World*, O´Reilly & Associates, Inc. 1995.

[11] Martin Kuba: *Web Services*, UVT-MU 2003.

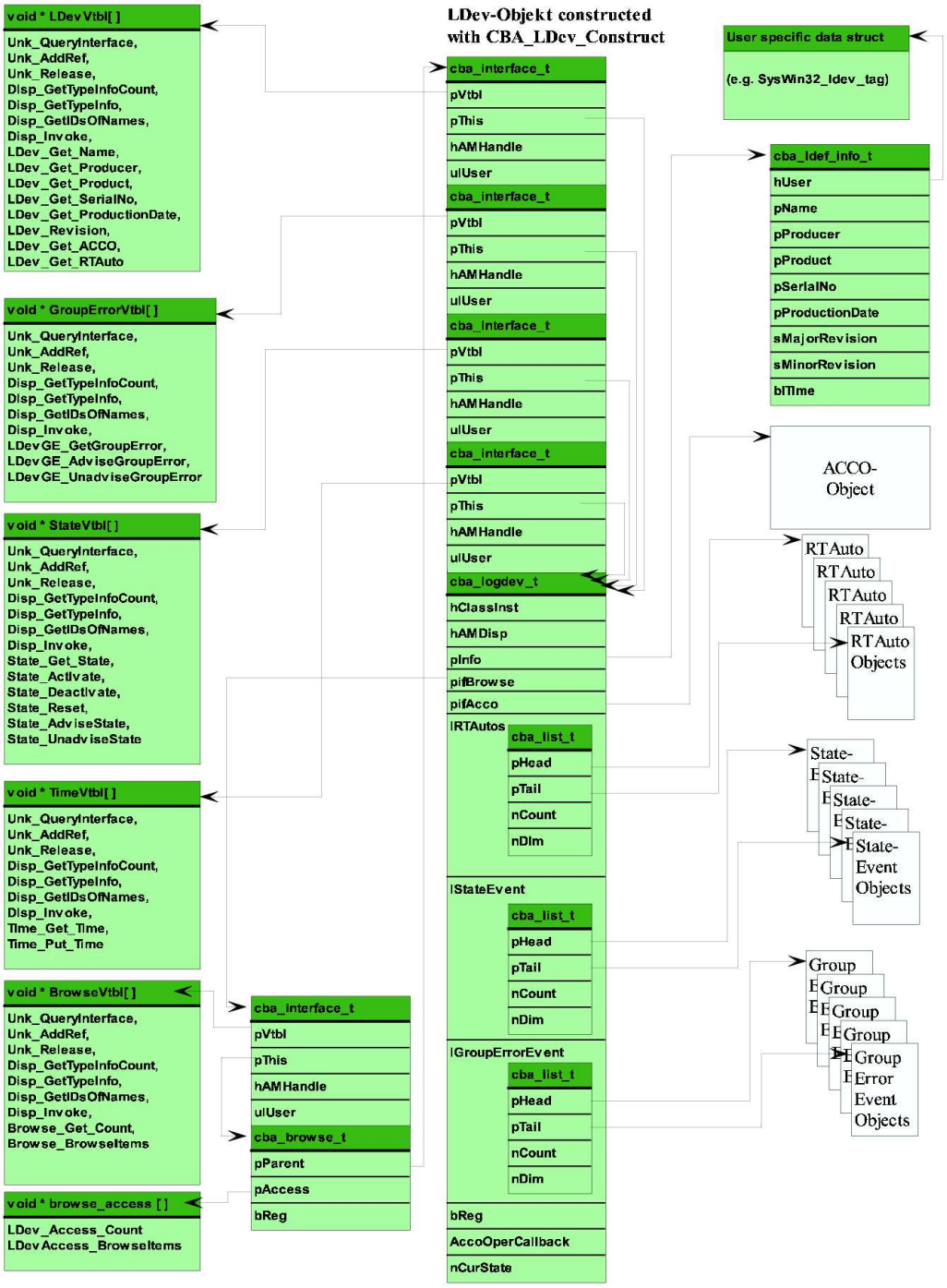
[12] Kelbel Jan: *Diplomová práce - Řídící systém chovu ryb*, ČVUT FEL, Praha 2004.

[13] Projekt *Apache*: URL: http://www.apache.org.

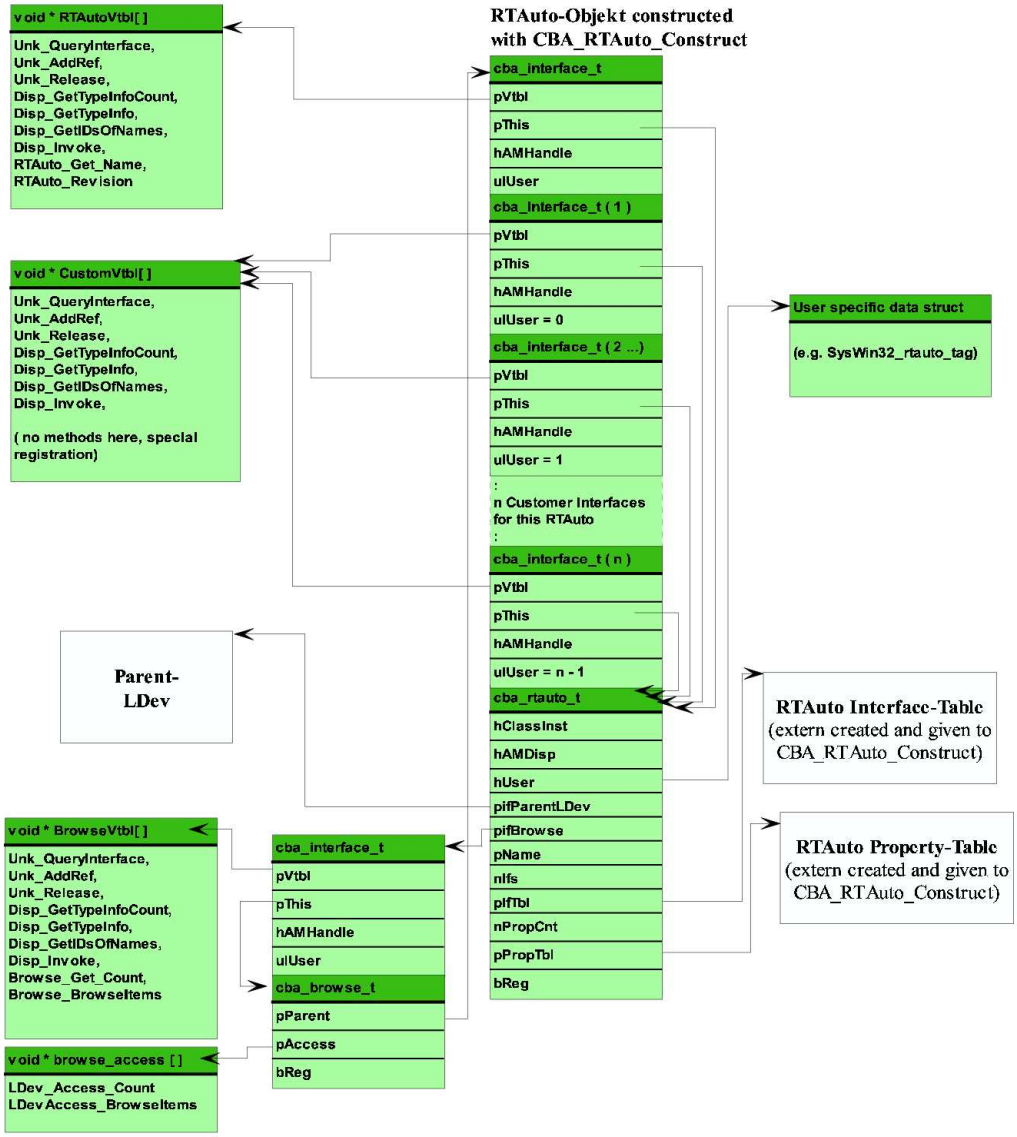
Přílohy



Obrázek 6: Příloha: Datová struktura objektu Physical Device



Obrázek 7: Příloha: Datová struktura objektu Logical Device



Obrázek 8: Příloha: Datová struktura objektu RTAuto

PDev Name:	lenee
Producer:	ifak e.V. Magdeburg
Product:	PROFInet4Linux
Serial No:	$Name: PNO_01-02-01-019_R_01 $
Date:	2003-10-2 21:30:39
Revision:	1.2
LDev List:	LDev_Counter LDev_Calculator LDev_DataTypeMirror

Obrázek 9: Příloha: Implicitní HTML stránky objektu Physical Device

LDev Name:	LDev_Calculator
Producer:	Logical Device
Product:	ifak e.V. Magdeburg
Serial No:	$Name: PNO_01-02-01-019_R_01 $
Revision:	1.2
Date:	2003-10-2 21:30:39
State:	Operating
Time:	2003-10- 2 21:42:26
Acco Ping Factor:	10
Group Error:	Ok
RTAuto List:	RTAuto_Calculator
Connection List:	

ID	Provider	Property	Conn. Name	Type	Value	ConnState	ConnChannel	QoS
2	192.168.0.66!LDev_Counter	CounterValue	RTAuto_Calculator.OperandA	I4	0	active	Local	100ms
3	192.168.0.66!LDev_Counter	CounterValue	RTAuto_Calculator.OperandB	I4	0	active	Local	100ms

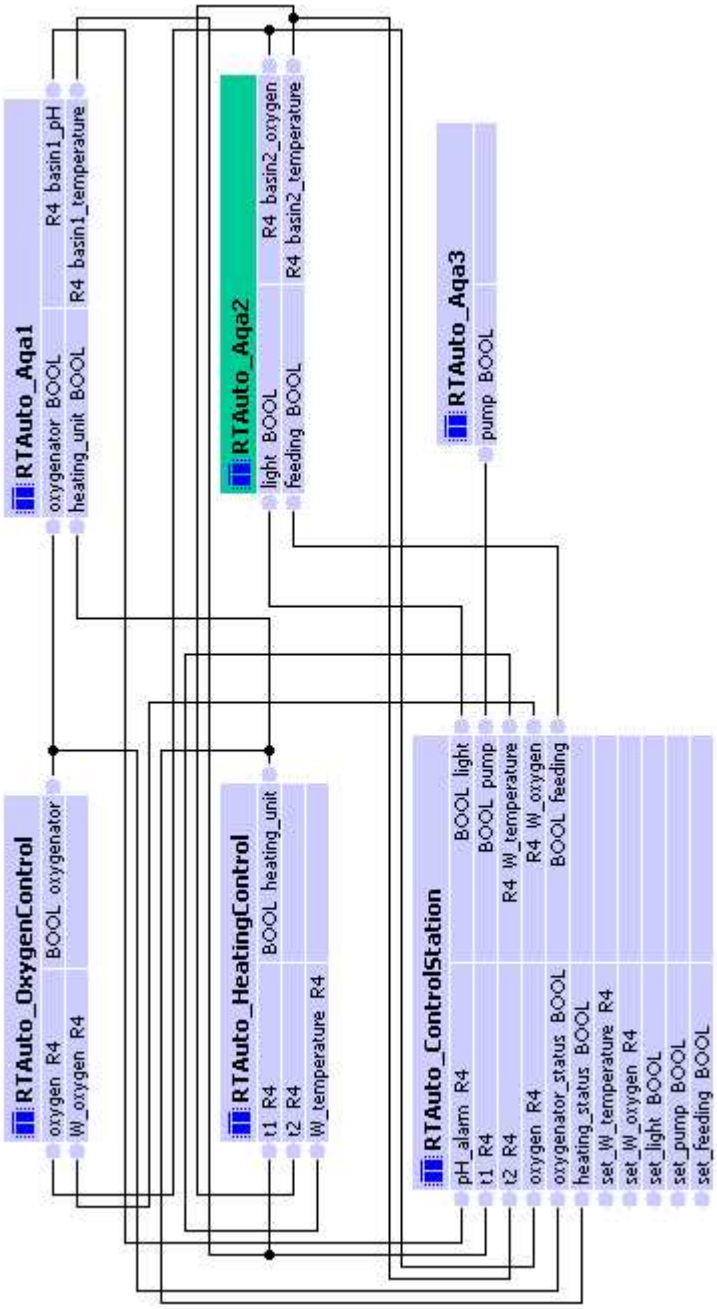
Obrázek 10: Příloha: Implicitní HTML stránky objektu Logical Device

	RTAuto_DataTypeMirror				
false	InVT_BOOL	BOOL	BOOL	OutVT_BOOL	false
0	InVT_I1	I1	I1	OutVT_I1	0
0	InVT_UI1	UI1	UI1	OutVT_UI1	0
0	InVT_I2	I2	I2	OutVT_I2	0
0	InVT_UI2	UI2	UI2	OutVT_UI2	0
0	InVT_I4	I4	I4	OutVT_I4	0
0	InVT_UI4	UI4	UI4	OutVT_UI4	0
0.000000	InVT_R4	R4	R4	OutVT_R4	0.000000
Input	InVT_BSTR	BSTR	BSTR	OutVT_BSTR	Output
	InVT_DATE	DATE	DATE	OutVT_DATE	

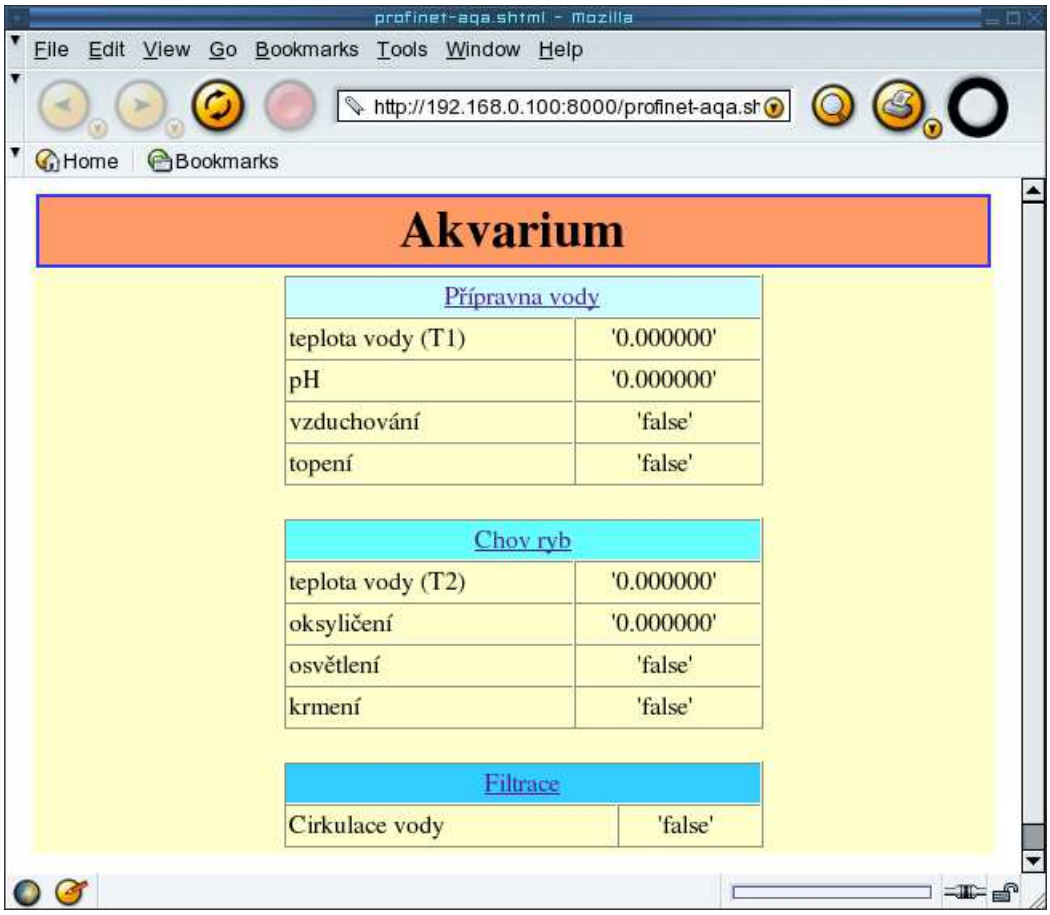
Obrázek 11: Příloha: Implicitní HTML stránky objektu RTAuto

RTAuto_DataTypeMirror.InVT_I4 0 PutValue

Obrázek 12: Příloha: Modifikace atributu objektu RTAuto



Obrázek 13: Příloha: Návrh propojení mezi objekty



Obrázek 14: Příloha: HTML stránka vytvořena pomocí SSI - komponenta Akvárium

Obrázek 15: Příloha: HTML stránka vytvořena pomocí SSI - komponenta Control

Obsah přiloženého CD

Adresář	**Popis**
./Aqaria.Pn/	zdrojové soubory komponenty Akvarium
./Aqaria.iMap/	komponenty pro iMap
./Aqaria_Control/	zdrojové soubory komponenty Control
./RS485/	zdrojové soubory ovladače RS485
./gen/	generované spustitelné soubory
./lib/	knihovy PROFInetu
./v1.2-src/	zdrojové soubory PROFInetu, přeložené *.o soubory jsou ukládány do adrseře ./lib
./cgi/	zdrojové soubory CGI skriptu pro webové rozhraní
./cgi-soap/	zdrojové soubory CGI skriptu pro poskytování webových služeb
./apache/	webový server Apache
./gSOAP-linux-2.3/	knihovny pro gSOAP (verze pro linux)
./gSOAP-win32-2.3/	knihovny pro gSOAP (verze pro win32)
./soap-client-linux/	ukázkový SOAP klient (verze pro linux)
./soap-client-win32/	ukázkový SOAP klient (verze pro win32)
./docs/	dokumentace
./docs/dp/	diplomnová práce v *.pdf* a *.ps* verzi
./AllInOne.tar.gz	vše zabalené v archivu zdůvodu zachovaní atributů jednotlivých souborů