TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: B2612 – Elektrotechnika a informatika
Studijní obor: 2612R011– Elektronické informační a řídící systémy

Vzdálený sběr a archivace dat z měřících přístrojů SMP

Remote collection and archiving of datas from SMP measuring devices

Bakalářská práce

Autor: **Martin Blížkovský**
Vedoucí práce: Ing. Jan Kraus
Konzultant: Ing. Tomáš Mikolanda

V Liberci 24.5.2009

ORIGINÁLNÍ ZADÁNÍ

ORIGINÁLNÍ ZADÁNÍ

ORIGINÁLNÍ ZADÁNÍ

ORIGINÁLNÍ ZADÁNÍ

ORIGINÁLNÍ ZADÁNÍ

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé BP a prohlašuji, že **s o u h l a s í m** s případným užitím mé bakalářské práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít své diplomové práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

Datum

Podpis

Abstrakt

Účelem projektu je navrhnout multiprocesový komunikační program, který bude schopen pomocí vhodného druhu meziprocesové komunikace přijímat příkazy, na jejichž základě bude stahovat data z vybraných měřících přístrojů SMP. Získaná data následně uloží do databáze nebo odešle aplikaci, která o ně požádala. Za tímto účelem je nutné seznámit se s různými typy meziprocesové komunikace v prostředí .NET a jejich možnostmi. Podle zadaných kritérií se vybere nejvhodnější typ této komunikace. Po zvolení způsobu komunikace se musí navrhnout protokol, pomocí něhož budou vybrané aplikace komunikovat. Nakonec, před návrhem samotné komunikační aplikace, je vhodné seznámit se s možnostmi tvorby multiprocesových aplikací v programovacím jazyce C# z vývojového prostředí Microsoft Visual Studio.

Po výběru vhodného způsobu komunikace a návrhu komunikačního protokolu byla postupně tvořena komunikační aplikace. Nejprve aplikace začala pracovat s jednoduchými funkcemi a příkazy, na jejichž základě se ověřila správnost navrženého protokolu. Po vylepšení drobných nedostatků v navrženém protokolu, objevených při testování na jednoduchých funkcích bylo možné vytvořit i složitější úkoly jako stahování záznamů. Posledním krokem návrhu bylo ukládání stažených dat do databáze.

Nyní je aplikace schopná přijímat příkazy od „libovolného" množství klientů, jejich požadavky najednou zpracovávat a na jejich základech jednat. Mezi její schopnosti patří několik funkcí pro stahování různých typů dat a také složitější algoritmy pro stahování archivních záznamů a jejich následné ukládání do databází.

Klíčová slova: Sběr dat, Ukládání dat, Meziprocesová komunikace, Multiprocesová aplikace, Komunikační protokol

Abstract

Main purpose of this project is design of multithreaded communication application which will be able to download datas from selected SMP devices by received orders through suitable inter-process communication. Downloaded datas will by then saved to database or send back to requesting application. For this purpose it is necessary to get knowledge about diferent kinds of inter-process communicaton and with their possibilities in .NET enviroment. According to specified criteria will be chosen best available kind of this communication. After selection of communication type its necessary to design protocol which will be used for communication between selected applications. Finally before designig of communication application its good to get familiar with possibilities of creating multithreaded application in C# programming language from Microsof Visual Studio programming enviroment.

After selection of suitable kind of communication and designing of communicatiom was gradually created communication application. At first the application was able to work with simple functions and orders. Rightness of designed protocol was then verifyed on them. After improvments of some minor issues in designed protocol which were discovered by testing on simple funcitons it was able to create more difficult tasks like downloading of archives. The last step of designig was saving downloaded datas into database.

Nowadays the application is able to receive orders from any number of clients, process their request at same time and act on their bases. There are couple of functions for downloading diferent types of records and also some advanced algorithms for downloading and saving of archived records.

Keywords: Data collecting, Data saving, Inter-process communication, Multithreaded application, Communication protocol

Obsah

1 Úvod

1.1 Vzdálený sběr dat

Pokud chceme znát a dále zpracovat například naměřené hodnoty z několika měřících stanic a nepoužijeme k tomu vzdálený sběr dat, musíme všechny měřící stanice obejít, hodnoty zaznamenat a až následně je můžeme začít vyhodnocovat a dále zpracovávat. Tento způsob je velice neefektivní, charakterizuje ho hlavně značná časová náročnost a potřeba pracovníků, kteří odečet provádějí. Negativa tohoto způsobu sběru dat navíc značně narůstají se zvyšujícím se počtem měřících stanic. V dnešní době se proto začínají zaznamenaná data z měřících stanic stahovat vzdáleně, pomocí výpočetní techniky. V praxi to znamená, že jsou všechny stanice ve spojení s jedním nebo více servery a ty obstarávají sběr dat. Systém může být ovládán minimálním množstvím pracovníků a načítání změřených hodnot může být v závislosti na použitých zařízeních i téměř okamžité. Další výhodou použití systému vzdáleného sběru dat je možnost měnit nastavení a konfigurovat jednotlivé měřící stanice například na základě vyhodnocených dat. Konfigurace může tak být provedena na dálku bez návštěvy konkrétní stanice. Zvolený způsob zisku dat nám nabízí značné množství dalších doplňujících funkcí. Jako další mohu jmenovat měření a zobrazování aktuálně měřených hodnot. Dochází tak k přesunutí zobrazovacích zařízení jednotlivých měřících přístrojů na libovolnou vzdálenost do předem určeného místa, kde je možné sledovat měřené hodnoty ze všech stanic najednou a na jejich základě pak jednat. Vedle všech vyjmenovaných výhod má ale systém i několik nevýhod. Hlavní nevýhodou je nutnost propojení veškerých zařízení do sítě se servery, které obstarávají přenos dat. Z tohoto důvodu není vzdálený sběr dat vhodný ve všech odvětvích používajících měření a záznam dat. Nejčastěji se tedy s tímto systémem setkáváme u měření elektrických veličin, u kterých může být propojení do sítě realizováno nejsnadněji.

1.2 Vzdáleně řízená aplikace

Ve většině případů vzdáleného sběru dat je nutné, aby byl server spojen s přístroji neustále. Není tedy nejvhodnější provádět zpracování naměřených hodnot přímo na něm, jelikož bychom se limitovali pouze na jedno vyhodnocovací pracoviště, které by muselo zastat všechny potřebné funkce. Proto je v některých případech

výhodnější provozovat na serveru vzdáleně řízenou aplikaci, která dostává příkazy z vyhodnocovacích pracovišť a následně provádí požadované činnosti. Zjednodušeně řečeno zprostředkovává komunikaci mezi vyhodnocovací a měřící stanicí. Aplikace však nemusí být nutně umístěna na vzdáleném serveru, muže být umístěna i na stejném počítači jako uživatelské rozhraní. Při tomto způsobu použití je funkcí aplikace oddělení programu pracujícího přímo s měřícími přístroji od grafického prostředí zajišťujícího zobrazení naměřených dat a možnosti různých nastavení, které uživatel využívá. Z uživatelského prostředí jsou předávány jednoduché zprávy vzdálené aplikaci, která je spuštěna na pozadí a do činnosti uživatel žádným způsobem nezasahuje. Aplikace přijme zprávu, vyhodnotí ji a na jejím základě vykoná uživatelem požadovanou činnost.

1.3 Použité prostředky

Hlavním cílem této bakalářské práce bylo navrhnout vzdáleně řízenou aplikaci pro komunikaci s měřícími přístroji SMP od společnosti KMB systems, s.r.o. a navrhnout vhodný komunikační protokol pro spojení s uživatelským prostředím. Aplikace byla navržena v programovacím jazyce C#. Jako programovací prostředí jsem z počátku používal Visual Studio 2005 s .NET 2.0 a v průběhu jsem přešel na novější verzi Visual Studio 2008 s .NET 3.5 od společnosti Microsoft a k němu určené rozšiřující moduly DXperience od společnosti Developer Express. Vzhledem k možnosti aplikace ukládat data z přístrojů do databáze, bylo navíc nutné použít program pro správu databází SQL Server 2008 taktéž od společnosti Microsoft.

2 Analýza požadavků pro stahování dat z přístrojů SMP

2.1 Způsob komunikace mezi měř. přístroji a vzdáleně řízenou aplikací (Service)

Komunikaci je možné rozdělit na dva základní způsoby. Prvním způsobem je spojení s přístrojem přes ethernet, u kterého je možné vzdálené spojení přes různé druhy sítí. Alternativou k ethernetu je v tomto případě spojení s měřícím přístrojem přes sériovou linku. Tento druh spojení je dále možné rozdělit podle používaného portu na USB a COM.

Pro navázání spojení s přístrojem je použit komunikační kanál *KMB.Communication.AbstractChannel* z knihoven KMB. Jedná se o „univerzální kanál", ke kterému je možné přiřadit libovolný komunikační kanál, potřebný pro spojení s etherentovými přístroji nebo s přístroji na sériové lince. Pro ethernetové se přířadí příkazem *KMB.Communication.TCPIO* a pro sériové příkazem *COMIO* ze stejné knihovny.

Přímé navázání spojení a výměna dat s přístrojem je v Service možné provést přímým zasláním pole bajtů představujících požadavek. Zpráva odesílaná přístroji je typu *MessageKmbLong* a před odesláním je její obsah vytvořen pomocí příslušného příkazu (viz Výpis 2.1). Po odeslání pomocí úkonu *communicate,* náležícího nadefinovanému kanálu, aplikace čeká na odpověď přístroje. Po přijetí odpověd uloží do *Listu* (seznam proměnných jednoho typu) zpráv *MessageKmbLong*. Pro další práci s přijatými daty je následně nutné převést přijaté bajty do struktury odpovídající typu zprávy.

```
List<MessageKmbLong> Ans = new List<MessageKmbLong>();
MessageKmbLong Ident = MessageKmbLong.getIdentify(devaddr);
comm.open();
Ans = comm.communicate(Ident);
comm.close();
```

Výpis 2.1: Ukázka způsobu komunikace s přístrojem

Druhou možností je použít funkci z knihoven KMB, která tyto úkony pro nejpoužívanější typy zpráv zajistí (viz Výpis 2.2). V případě, že se jedná o dotaz bez zasílání doplňujících dat, předáme jí jako paramter pouze typ dotazu, komunikační kanál, který má být použit a adresu přístroje na dané lince, výstupem je po dokončení komunikace přímo požadovaná struktura. V jiných případech, kdy jsou přístroji předávána doplňující data, je možné použít jinou odpovídající funkci.

```
KmbStructure Struct = null;
comm.open();
Struct = ServiceFacade.instance.getSmpData(typ, comm, devaddr);
comm.close();
```

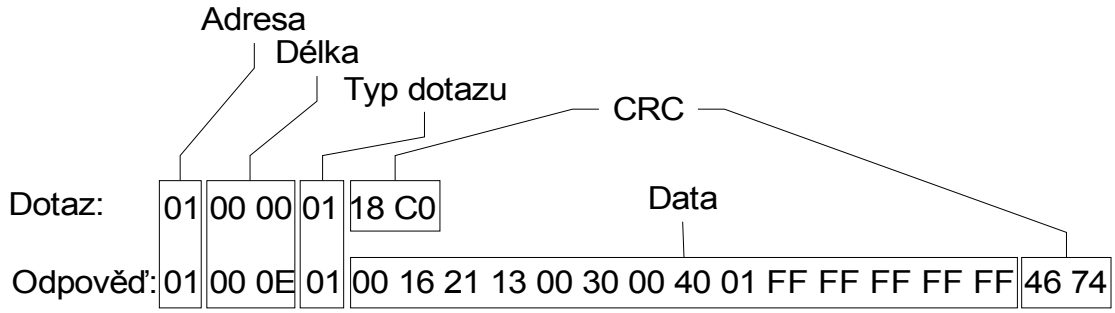
Výpis 2.2: Ukázka způsobu komunikace s přístrojem

2.2 Rozbor zpráv zasílaných mezi měřícími přístroji a Service

Zpráva zasílaná měřícímu přístroji má podobu několika bajtů. Každý bajt je v hexadecimálním tvaru. Jednotlivé zprávy jsou rozděleny na úseky bajtů podle jejich funkce (viz Obrázek 2.1). Prvním bajtem zprávy je adresa. Jedná se o adresu přístroje na dané komunikační lince. Jelikož místo pro adresu má kapacitu jeden bajt, může být na jedné lince až 255 přístrojů s adresami od 1 do 255. Dva bajty následující po adrese přístroje udávají velikost dat ve zprávě. Zobrazují počet bajtů představujících data pro nebo od přístroje. Dalším bajtem se rozlišuje typ požadavku zasílané zprávy. Následuje řada bajtů představujících data. Jejich počet, jak již bylo zmíněno je obsažen v jednom z předcházejících bajtů. Každá zpráva je zakončena dvojicí bajtů představujících CRC. Před každým odesláním zprávy je CRC pro danou zprávu vypočítáno a zařazeno na její konec. Při přijetí zprávy je CRC obsažené ve zprávě porovnáno s CRC, které se nově ze zprávy vypočítá. V případě neshody je okamžitě zřejmé, že při přenosu zprávy došlo k chybě a přijatá zpráva s největší pravděpodobností neodpovídá odeslané.

Na obrázku 2.1 je zobrazen dotaz, který byl zaslán přístroji a odpověď na tento dotaz obdržená od přístroje. Typem dotazu je v této ukázce identifikace, po jejímž obdržení přístroj odešle zpět svoje identifikační údaje. Jedná se o jeden z nejjednodušších typů dotazu, který přístroji nezasílá ve zprávě žádná data, což je vidět

na dvou bajtech určujících délku dotazu. U odpovědi je na pozici pro délku hodnota 00 0E v hex. Po převedení do decimální soustavy vyjde 14, což odpovídá počtu bajtů v kolonce Data.



Obr. 2.1: Popis zpávy

2.3 Speciální případy výměny zpráv mezi přístroji a Service

Běžným způsobem komunikace je obdržení jedné odpovědi na jeden odeslaný dotaz. Při sledování komunikace s měřícími přístroji jsem však zjistil, že v některých případech je nutné na jeden dotaz zaslat zpět více než jednu odpověď. Například pří stahování archivních záznamů je nutné na pokyn jednoho dotazu odpovědět více zprávami v závislosti na velikosti daných záznamů. Jiným případem je stahování aktuálních naměřených dat z přístroje, aplikace v tomto případě musí zajistit opakované stahování hodnot z přístroje po určitých časových okamžicích a zasílat je zpět klientské aplikaci pro další zpracování. Při návrhu aplikace Service bylo nutné s těmito případy počítat a zajistit vhodným návrhem funkcí jejich správné plnění.

Dalšími případy jsou dotazy, na které je odpověď zaslána přímo odesílateli a dotazy, jejichž odpověď je zpracována, případně uložena bez další interakce s klientskou aplikací. Navržená aplikace musí tedy zajistit, aby archivní záznamy nebyly zasílány zpět klientské aplikaci, ale uloženy do databáze. Na druhou stranu aktuální naměřené hodnoty musí být zasílány zpět tazateli, jinak by postrádaly svůj smysl.

3 Komunikace

3.1 Meziprocesová komunikace v .NET

Meziprocesová komunikace (Inter-Process Communication) slouží ke komunikaci mezi jednotlivými vlákny aplikace. Komunikace nemusí probíhat pouze mezi vlákny jedné aplikace, ale i mezi různými aplikacemi, jejich umístění není omezeno prostředím jednoho počítače, jelikož za pomoci některých druhů meziprocesové komunikace mohou komunikovat i aplikace na různých počítačích spojených v síti. Různé způsoby meziprocesové komunikace jsem rozdělil za pomoci [6] a [7] na Vzdálené volání procedur (Remote procedure calls), Synchronizace, Sdílená paměť(Shared memory) a Předávání zpráv (Message passing).

3.1.1 Vzdálené volání procedur

Používá se, jak již název napovídá, ke vzdálenému volání procedur. Využití najdeme například při použití jedné nebo více aplikací, sdílejících jednu funkci (proceduru). V takovém případě může být aplikace obsahující volané funkce umístěna na nějakém serveru, ostatní aplikace využívající tyto funkce jsou propojeny se serverem sítí a dle potřeby zasílají serveru požadavky o vykonání zmíněných funkcí. Tento způsob je možné používat například při provádění složitějších matematických výpočtů, uživatelské prostředí pro zadávání parametrů počítané úlohy je umístěno na jednom počítači, po zadání je zavolána pomocí vzdáleného volání procedur funkce, která samotný výpočet úlohy provede. Funkce může být umístěna na výkonném počítači, sloužícímu pouze k vykonávání požadovaných výpočtů z jiných počítačů.

3.1.2 Synchronizace

V případech, kdy je nutné dodržet určitý časový sled činností jednotlivých procesů, můžeme použít synchronizaci. Procesy si při dosažení předem určených podmínek v programu předají informaci o tomto stavu, na jejich základě pak v případě nutnosti mohou přizpůsobit svoji činnost. Tímto způsobem procesy dokončí úkol v předem požadovaném sledu a nestane se tak například případ, kdy proces závisející na výsledku jiného procesu skončí dříve než bude mít onen výsledek k dispozici. Pro synchronizaci se nejčastěji používají metody *signál*, *semafor* nebo *bariéra*.

3.1.3 Sdílená paměť

Meziprocesová komunikace pracující na jednom počítači mezi více aplikacemi. Jedna aplikace si v operační paměti počítače (RAM) vyhradí prostor, do kterého mají přístup díky meziprocesové komunikaci i další pověřené aplikace a mohou si tak díky vymezenému prostoru mezi sebou vyměňovat informace. Vzhledem k výměně informací přímo přes operační paměť se jedná o jeden z nejrychlejších způsobů komunikace mezi aplikacemi. Způsob komunikace, díky kterému dosahuje vysoké rychlosti, však brání použití tohoto způsobu předávání dat ke komunikace mezi počítači spojenými v síti. Další podmínkou je, což ale platí pro většinu druhů meziprocesové komunikace, nutnost současného běhu aplikací sdílejících vyhrazený prostor v operační paměti. Proto se její použití musí omezit pouze na aplikace, které jsou spuštěny současně a pouze na jednom počítači.

3.1.4 Předávání zpráv

Procesy mají možnost zasílat ostatním celé zprávy. Metody je možné rozdělit na asynchronní a synchronní. V případě asynchronních příjemce nemusí neustále naslouchat a očekávat zprávu. Zprávy jsou odesílatelem odeslány a následně uloženy v závislosti na použité metodě na některém z počítačů, kde jsou aplikace spuštěny, příjemce si po té může zprávy vyzvednout téměř kdykoliv. U synchronních metod musí příjemce při odesílání naslouchat a okamžitě odesílaná data přijímat, proces přenosu zprávy je realizován přímým spojením odesílatele s příjemcem. Hlavními metodami, které se řadí mezi předávání zpráv jsou socket, pipe, named pipe, message queue nebo soubor.

Socket

Hlavním účelem této metody pro předávání zpráv je jejich zasílání přes síť, ale je možné ji použít i v rámci jednoho počítače. Nejčastěji se ke komunikaci pomocí této metody využívá komunikační protokol TCP, ale je možné použít například i UDP. Procesy při komunikaci nepracují na stejné úrovni. Jeden plní roli serveru a pouze čeká na příchozí spojení a následně přijímá data, je tedy pasivní. Druhý je klientem, který vytváří spojení a zasílá zprávy serveru, je aktivní. Při použití pro komunikaci mezi více vlákny nebo procesy si každé vlákno musí vytvořit vlastní spojení. Zde můžeme narazit

na problém v případě použití většího množství komunikujících vláken. Každé vlákno si musí vytvořit vlastní komunikační kanál, obstarat komunikaci a nakonec korektně uzavřít vytvořené spojení. Počet komunikačních kanálu je tedy roven počtu vláken programu použitých ke komunikaci, což může být v některých případech velice náročné.

Pipe(Roura)

Jak již její název při překladu do českého jazyka napovídá, přenos informací touto metodou probíhá obdobně jako v rouře. Zprávy postupují jako ve frontě (FIFO), nebo-li první odeslaná zpráva je na druhém konci také první doručena. Při jejím vytváření v paměti počítače vznikne jednosměrný proud dat (roura) od jednoho spuštěného procesu k druhému. Po ukončení procesů pipe automaticky zanikne. Pipe je v systému charakterizována čtecím a zapisovacím koncem. Pokud proces Pipe vytvoří, vrátí mu informace o obou jejích koncích. Získanou informaci si procesy při komunikaci předávají. Primární využití této metody je pro komunikaci mezi procesy na jednom počítači. Pro komunikaci mezi procesy na počítačích v síti je nutné vytvořit Pipe s pomocí spojení TCP (socket) , z čehož plyne, že pro vzdálenou komunikace je výhodnější použít přímo TCP spojení místo Pipe.

Named Pipe

Pracuje na téměř stejném principu jako Pipe. Hlavním rozdílem mezi těmito metodami je rozdílný způsob zisku informace o koncích pipe. V tomto případě se při vytvoření informace uloží do souboru, ze kterého si komunikující procesy podrobnosti o pipe přečtou a na jejich základě komunikují.

Message queue

Jedná se o frontu zpráv, možnou použít jak pro komunikaci mezi procesy na jednom počítači, tak i pro vzdálenou komunikaci. Data jsou ve frontě předávána ve formě zpráv, proto název Message queue. Fronta může být v počítači vytvořena trvale, nezávisle na běhu komunikujících procesů. Po té, co proces předá odesílanou zprávu do fronty, je zpráva uložena ve frontě do té doby dokud si jí některá z aplikací podílející se na komunikaci nevyzvedne, předávají se tedy asynchronně. Celá fronta i všechny

zprávy, které obsahuje zůstanou v systému počítače uloženy dokonce i po jeho vypnutí. Primárně se jedná o jednosměrnou komunikaci, jelikož proces žádným jednoduchým způsobem nepoznají, zda je ve frontě zpráva, kterou odeslal nebo zpráva, kterou má přijmout. Navíc, pokud je ve frontě více zpráv, čtení začíná od zprávy, která je ve frontě uložená nejdéle. Obousměrná komunikace ve frontě je možná, jelikož oba procesy mají stejné možnosti v přístupu do fronty, ať už se jedná o odesílatele nebo příjemce, ale při předávání zpráv by bylo nutné zajistit vhodně synchronizaci obou aplikací a podmínky, za kterých mohou zprávy odesílat. Nejvýhodnějším způsobem obousměrné komunikace při použití message queue je vytvoření dvou front mezi dvěma procesy. I pro multiprocesové aplikace nám díky asynchronnímu přenosu stačí pouze jedna fronta pro jeden směr komunikace, což je značná výhoda oproti komunikaci za použití metody socket. V prostředí .NET tuto metodu využívá technologie s názvem Microsoft Message Queuing (MSMQ).

Microsoft Message Queuing

Technologie, zprostředkovávající v prostředí .NET meziprocesovou komunikaci pomocí metody message queue. MSMQ je podle [9] zahrnuta v operačním systému Windows již od systému Windows 95, ale pouze ve formě instalačního balíčku a dříve než se použije, musí se nainstalovat. MSMQ zaručuje doručení odeslaných zpráv příjemci také díky schopnosti uložit ve frontě zprávy do doby než bude příjemce dostupný a z fronty si je vyzvedne. Komunikaci je možné provádět přes různé druhy sítí. Jednotlivým zprávám je možné před odesláním nastavit různé priority. Existuje již několik verzí MSMQ, které odpovídají použitému operačnímu systému. Jak je uvedeno v [9] první verze MSMQ 1.0 byla dostupná v operačních systémech Windows NT4, Windows 95, Windows 98, MSMQ 2.0 byla zahrnuta v systému Windows 2000, MSMQ 3.0 odpovídá systému Windows XP a Windows 2003, MSMQ 4.0 je obsažena ve Windows Vista a Windows 2008. Jednotlivé verze MSMQ jsou zpětně kompatibilní, což znamená, že každá novější verze podporuje funkce všech starších verzí. Pokud je nutné přejít na vyšší verzi MSMQ je nutné nainstalovat novější verzi operačního systému, jelikož verze MSMQ je dána operačním systémem. Ve Windows je také možnost MSMQ spravovat. K této činnosti slouží nástroj s názvem Řízení front zpráv. Zde je možné prohlížet, mazat a nastavovat fronty zpráv, které již byly v počítači vytvořeny nebo vytvářet nové, prohlížet zprávy, které jsou momentálně ve frontě uloženy. Nachází

se zde možnost náhledu obsahu jednotlivých zpráv a jejich doplňujícího nastavení. Případně je zde také možnost doinstalovat další doplňující funkce k MSMQ.

3.2 Protokol zprávy

Po výběru vhodného komunikačního protokolu pro meziprocesovou komunikaci mezi službou a uživatelským prostředím bylo nutné vymyslet vhodný formát zpráv, který bude data přenášet. Hlavním požadavkem na formát zprávy byla možnost zprávu přečíst a případně jí i porozumět bez speciálního softwaru na její zpracování. V kombinaci s použitým programovacím jazykem bylo nejvýhodnější použít pro formát zprávy Extensible Markup Language spíše známy pod zkratkou XML. Použitý jazyk je mezinárodní a standardizovaný organizací W3C. V současnosti je nejčastěji využíván k přenosu dat, nebo ke zpracování různých textových dokumentů. Jednou z jeho největších výhod je standardizovanost, není tedy nutné k jeho přečtení používat jeden konkrétní software, ve kterém byl například dokument vytvořen, ale můžeme ho přečíst v téměř každém běžném textovém editoru. Další výhodou jsou XML značky (tagy). Pomocí nich jsou jednotlivé části textu rozděleny do bloků, kde každá značka může značit význam daného bloku. Pro člověka tak není příliš složité obsahu alespoň z části porozumět i bez znalosti konkrétních souvislostí.

Zprávu jsem se rozhodl rozdělit do dvou hlavních částí (viz Výpis 3.1). První je hlavička nazvaná *MsgHead*. Hlavička obsahuje důležité informace o každé jednotlivé zprávě. Určuje kdo a odkud zprávu odeslal. Konkrétně se jedná o *ClientAdress* neboli IP adresu odesílatele zprávy a také *ClientQName*, což je název fronty vytvořené konkrétním klientem, který zprávu odeslal ke komunikaci s danou aplikací (Service). Dále je v hlavičce zahrnuto, komu je zpráva určena, *ServiceAdress* určující IP adresu Service, se kterým klient komunikuje. Jednotlivé zprávy bylo nutné od sebe rozlišovat. Hlavním důvodem byla následovná možnost identifikace zprávy. Jelikož hlavička zprávy, kterou klient odešle se shoduje s hlavičkou zprávy s odpovědí na klientův dotaz, klient musí poznat, na který dotaz obdržel odpověď. Z toho důvodu jsem do zprávy zahrnul *MsgID*. Jedná se o jakési identifikační číslo a každý pár zpráv dotaz-odpověď obdrží vlastní, na jeho základě je pak klient schopen zprávy, při obdržení více odpovědí, spolehlivě rozlišit. Service, který zprávy vyhodnocuje a na jejich základě jedná, dělí

jednotlivé dotazy podle položky *Request*. Jde o stručný identifikátor, díky kterému určí typ dotazu a začne podle něj jednat. Většina zpráv obdržená Service, požaduje spojení a získání požadovaných dat z měřících přístrojů, jelikož přístrojů může být více, je nutné, aby bylo možné z hlavičky zprávy rozeznat, se kterým přístrojem navázat spojení. Service má v sobě uložen seznam dostupných přístrojů, se kterými může v daném okamžiku komunikovat. Zpráva obsahuje položku *Device*, která označuje klientem zvolený přístroj ke komunikaci. Service pak na základě již zmíněného seznamu určí přístroj k navázání spojení. Poslední věcí v hlavičce je *ProtocolVersion*. Ta značí verzi protokolu zprávy, za jehož pomoci byla zpráva vytvořena. Slouží k identifikaci dostupných možností dané verze protokolu při problémech s kompatibilitou jeho různých verzí.

Druhou částí zprávy je *MsgBody*. *MsgBody* vyjadřuje vlastní tělo zprávy, mohou zde být uloženy různé struktury dat v závislosti na požadavcích jednotlivých aplikací. Klient do těla zprávy umisťuje různá nastavení a doplňující informace k danému požadavku, u nejjednodušších požadavků však tělo zprávy může být úplně prázdné. Na druhou stranu, do těla většiny zpráv přicházejících od Service jsou data uložena ve formě předem nadefinovaných struktur. Nejčastěji se jedná o naměřená data, doplňující informace a nastavení měřících přístrojů, nebo informativní zprávy o stavu některých činností.

Celý protokol je v prostředí Visual Studio vytvořen jako samostatný projekt, který vytváří dll knihovnu. Vytvořenou knihovnu je možné v případě nutnosti přidat do dalších projektů, které funkce protokolu využívají. Výhodná je také snadná možnost rozšíření protokolu. Budoucí změny podoby hlavičky zprávy nebudou nejspíše nutné, ale jsou možné. Hlavní, je ale možnost změn těla zprávy. S dalším postupným vývojem aplikací využívajících navržený protokol, bude téměř jistě nutné zahrnout do těla zprávy další struktury pro ukládání dat do zprávy. Díky uvedenému návrhu a použití XML to nebude činit žádné větší problémy a za určitých podmínek, pokud nebude nutné zásadním způsobem měnit již zahrnuté struktury a formáty, může být zachována i zpětná kompatibilita protokolu.

```
<?xml version="1.0" encoding="utf-8"?>
<Message xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <MessageHead>
    <ProtocolVersion>1.0</ProtocolVersion>
    <ClientAddress>127.0.0.1</ClientAddress>
    <ClientQName>clientQ_0</ClientQName>
    <ServiceAddress>127.0.0.1</ServiceAddress>
    <MsgID>2</MsgID>
    <Request>IDENTIFY</Request>
    <Device>1</Device>
  </MessageHead>
  <MessageBody>
    <SmpIdentify>
                    …
    </SmpIdentify>
    <ListAvailableDevices />
  </MessageBody>
</Message>
```

Výpis 3.1: Ukázka zprávy ve formátu XML

4 Realizace komunikační aplikace (Service)

4.1 Vytvoření fronty zpráv s MSMQ

Při výběru vhodného druhu meziprocesové komunikace jsem se rozhodoval hlavně mezi třemi způsoby předávání zpráv, kterými byly *Named Pipes, Socket* a *MSMQ*. Jelikož aplikace měla být schopna zajišťovat i vzdálenou komunikaci zavrhl jsem použití technologie *pipes*, neboť jejich primární použití je pro lokální komunikaci. Ze zbývajících dvou jsem se rozhodl použít MSMQ. Důvodem této volby jsou problémy při použití *socketů* pro meziprocesovou komunikaci, pro každý nově spuštěný proces by bylo nutné vytvořit komunikační kanál s vlastním portem. Tento způsob by byl velice složitý a náročný,

Podpora pro technologii *MSMQ* je v systému Windows XP obsažena, ale není nainstalována. Prvním krokem byla tedy instalace *MSMQ*. Nainstalovat *MSMQ* je možné přes volbu *Přidat nebo odebrat součásti systému* v *Ovládacích panelech*. V mém případě jsem pro možnost budoucího využití použil automatický instalátor, který tyto úkony vykoná automaticky. Jedná se o spouštěcí soubor .bat a konfigurační soubor .ini. Spouštěcí soubor spouští systémový soubor *sysocmgr.exe* s určitými parametry (viz Výpis 4.1.1). V konfiguračním souboru se pak nachází nastavení instalované aplikace, v tomto případě *MSMQ* (viz Výpis 4.1.2).

```
sysocmgr.exe /i:sysoc.inf /r /u:config_in.ini
```

Výpis 4.1.1: Obsah instalačního .bat souboru

```
[Version]
Signature = "$Windows NT$"
[Global]
FreshMode = Custom
MaintanenceMode = RemoveAll
UpgradeMode = UpgradeOnly
[Components]
msmq_Core = ON
```

```
msmq_LocalStorage = ON
msmq_HTTPSupport = OFF
msmq_TriggersService = ON
msmq_ADIntegrated = ON
msmq_MQDSService = OFF
msmq_RoutingSupport = OFF
[msmq]
```

Výpis 4.1.2: Konfigurační soubor

V programovacím prostředí Visual Studio jsem musel k projektu serverovské aplikace *(Service)* přidat referenci S*ystem.Messaging*, která zpřístupní potřebné prostředky k použití *MSMQ*. V hlavní smyčce Service, která se spustí ihned po startu jsem vytvořil pomocí příkazu *MessageQueue.Create* (viz 4.1.3) frontu, do které klientské aplikace zasílají své dotazy. S použitím podmínky *if* a příkazu *MessageQueue.Exist* dochází před vytvořením fronty ke kontrole, zda už fronta v počítači neexistuje. V případě, že je nalezena již existující fronta, Service se k ní připojí. Při vytváření nebo hledání fronty se zadává umístění fronty ve formátu *string*. „Tečka" značí, že se jedná o lokální počítač, jako alternativu je možné místo ní zadat název počítače. *Private* udává soukromou frontu zpráv. Poslední částí je název fronty, v tomto případě *SeviceQ*. Příjem zpráv od klienta je realizován pomocí *ReceiveCompletedEventHandler,* u události *ReceiveCompleted* náležící přidané frontě (viz Výpis 4.1.4). Tato událost je spuštěna v okamžiku, kdy je zpráva stažena z fronty do aplikace. Stažení zprávy je realizováno pomocí funkce *BeginReceive*, která po jejím spuštění sleduje zadanou frontu zpráv a čeká na zprávu, která do fronty dorazí. Ve chvíli, kdy se zpráva ve frontě objeví, je spuštěna událost, která zavolá funkci *Received* (viz Výpis 4.1.5).

```
if (MessageQueue.Exists(@".\Private$\ServiceQ"))
        rec_Q = new System.Messaging.MessageQueue(@".\Private$\ServiceQ");
    else
        rec_Q = System.Messaging.MessageQueue.Create(@".\Private$\ServiceQ");
```

Výpis 4.1.3: Vytvoření fronty zpráv

```
rec_Q.ReceiveCompleted += new ReceiveCompletedEventHandler(Received);
rec_Q.BeginReceive();
```

Výpis 4.1.4: Příjem zprávy

```
private static void Received(Object Q, ReceiveCompletedEventArgs Arg)
{
  try
    {
      System.Messaging.MessageQueue mq = (System.Messaging.MessageQueue)Q;
      System.Messaging.Message m = mq.EndReceive(Arg.AsyncResult);
      mq.Formatter = new XmlMessageFormatter(new Type[] { typeof(String) });
      mq.BeginReceive();
      Msg Mes = new Msg();
      Mes = (Msg)x.Deserialize(m.BodyStream);
      Work(Mes);
    }
      catch (MessageQueueException ex)
      ...
}
```

Výpis 4.1.5: Funkce zajišťující dokončení příjmu zprávy

Po zavolání funkce *Received* pomocí uvedené události dojde k vytvoření nového vlákna aplikace, čímž se ze Service stává multi-procesová aplikace. Funkce obdrží po jejím spuštění dva parametry. Parametr *Q* typu *Object* představuje frontu zpráv, do které dorazila zpráva. Druhým je argument události *ReceiveCompleted,* pomocí něhož se následně dokončí proces příjmu zprávy. Ve funkci je na základě parametru *Q* vytvořena fronta a dále také proměnná typu *Message*, do které se uloží přijatá zpráva. Příjem je dokončen pomocí funkce *EndReceive,* s výše zmiňovaným argumentem. Po přijetí zprávy je nastaveno formátování zpráv ve frontě, v tomto případě *XML*. Před začátkem zpracování přijaté zprávy je opět spuštěna funkce *BeginReceive*, která čeká na další zprávu doručenou do fronty. Přijatá zpráva je typu *stream* a ve formátu *XML*. Zpráva je po té deserializována a uložena do mnou vytvořené třídy *Msg*, která je rozdělena na hlavičku *MsgHead* a tělo *MsgBody* zprávy (viz 4.1.6). Po načtení proměnných z *XML streamu* je zavolána funkce, třídící jednotlivé zprávy podle jejich dotazu (viz 4.2.1).

```
[XmlRoot("Message")]
public class Msg
{
   [XmlElement("MessageHead")]
   public MsgHead hd = new MsgHead();
   [XmlElement("MessageBody")]
   public MsgBody bd = new MsgBody();
}
```

Výpis 4.1.6: Třída Msg

4.2 Třídění zpráv podle dotazu klientské aplikace

Každá zpráva přijatá Service má v hlavičce *MsgHead* položku s označením *Request*. Na toto místo uloží klientská aplikace před odesláním zprávy svůj dotaz ve formě předem daného řetězce. Zprávy jsou v Service po přijetí rozděleny do dvou základních skupin.

Do první skupiny jsou zařazeny zprávy, které nepokračují dále přes Service k měřícím přístrojům, jelikož odpověď na ně je vytvořena přímo v Service. Ve většině případů se jedná o zprávy, zabývající se správou seznamu o dostupných měřících přístrojích, jako například nalezení nových přístrojů nebo jejich přidávání a odebírání ze seznamu uloženém v Service.

Druhou skupinou zpráv jsou ty, které požadují po Service zprostředkování komunikace s měřícím přístrojem. Service se po jejich obdržení spojí s přístrojem a vykoná úkoly, které má předem zadané podle typu požadavku obsaženém ve zprávě.

V programu je systém třídění zpráv vyřešen jako samostatná funkce, která je spuštěna po přijetí zprávy od klienta. Třídící algoritmus je sestaven z jednoduchých podmínek *if* a *else* (viz Výpis 4.2.1). V podmínkách se porovnává řetězec *Request*, ve výpisu *Mes.hd.Req* z hlavičky zprávy s předem nadefinovanými řetězci. Program postupně prochází jednotlivé podmínky do té doby, než je podmínka vyhodnocena pozitivně. V tuto chvíli je zavolána funkce náležící ke klientově požadavku. Pokud při porovnávání nedojde ani v jednom z případů ke shodě, dojde k odeslání chybové zprávy o neznámém typu požadavku zpět ke klientovi.

```
public static void Work(Msg Mes) {
    if (Mes.hd.Req == "GETDEVICES") GetAllDev(Mes);
    else if (Mes.hd.Device != 0) {
      try {
        AvailableDevices AD = AvDevL[Mes.hd.Device - 1];
      }
      catch {
        Ms.bd.ServiceMessage = "Device not found";
        return;
      }
      if (Mes.hd.Req == "IDENTIFY") Identify(Mes);
      else if (Mes.hd.Req == "GETCONFIG") GetConfig(Mes);
      else {
        Ms.bd.ServiceMessage = "This kind of request is not available";
        return;
      }
    }
    else if (Mes.hd.Device == 0) {
      Ms.bd.ServiceMessage = "Device not selected";
      return;
    }
    else {
      Ms.bd.ServiceMessage = "This kind of request is not available";
      return;
    }
  }
```

Výpis 4.2.1: Ukázka části funkce pro třídění zpráv

4.3 Synchronizace a priority operací pracujících s měřícími přístroji

Měřící přístroje SMP připojené přes COM a TCP, se kterými pracuje moje aplikace mohou v jeden okamžik přijímat pouze jeden dotaz. Jelikož zbytek komunikačního řetězce (klienti a Service) může pracovat ve více vláknech (multi-threading) bylo nutné zajistit přechod z více vláken do jednoho pro každý konkrétní měřící přístroj.

Pro zajištění této funkce jsem v programu zavedl pole globálních proměnných typu *boolean* s názvem *Link_available*. Po startu programu se všechny položky pole

nastaví na hodnotu *true*, což znamená, že všechny komunikační linky jsou dostupné pro komunikaci. Před tím, než aplikace zahájí komunikaci s některým z měřících přístrojů, ověří pomocí podmínky *if*, zda je zvolená linka dostupná (viz Výpis 4.3.1), pokud je linka nedostupná vlákno programu zůstane zacyklené v čekací smyčce vytvořené pomocí podmínky *while* (viz Výpis 4.3.1) do doby, než dojde k jejímu uvolnění. Ve chvíli, kdy je linka dostupná se aktivní vlákno Service pokusí spojit s měřícím přístrojem a nastaví proměnou *Link_available* na hodnotu *false*. Po ukončení komunikace daného vlákna s přístrojem dojde k opačnému procesu, tedy hodnota *Link_available* se nastaví zpět na *true*. Takto popsaný princip, který zajišťuje komunikaci s přístrojem, by efektivně fungoval pouze pro jeden přístroj. Service je ale schopný komunikovat i s více přístroji a z toho důvodu je proměnná *link_available* deklarována jako pole. Při práci s tímto polem aplikace používá jako jeho indexy čísla, která jsou přidělena jednotlivým dostupným přístrojům. Tato čísla jsou známá i klientským aplikacím a proto se používá přímo označení přístroje z hlavičky zprávy obdržené od klienta. V ukázce funkce je toto označení přístroje skryto pod proměnnou *Mes.hd.Device*. Příkaz *lock*, který uzavírá celou uvedenou funkci, zde zajišťuje, aby byl kód programu „thread safe". Jeho funkcí je dočasné zamknutí proměnných v jednom vláknu a brání tak jejich změnám z jiných vláken. Vlákna, která chtějí změny provést, musí počkat do doby než budou proměnné opět odemčeny. *Link_Lock* je objekt, na kterém se ověřuje, zda je zamčen nebo odemčen. Proměnných *Link_Lock* je vytvořeno pole, kde každá položka slouží jednomu přístroji.

```
lock(link_lock[Mes.hd.Device]){
while (!link_available[Mes.hd.Device])
        { System.Threading.Thread.Sleep(wait); }
        if (link_available[Mes.hd.Device])
        {
           link_available[Mes.hd.Device] = false;
           comm.open();
           ElmConf = comm.communicate(ElmConfmsg);
           comm.close();
        }
        link_available[Mes.hd.Device] = true;
}
```

Výpis 4.3.1: Ukázka použití Link_available

Dalším algoritmem pro zajištění komunikace mezi Service a měřícími přístroji je systém různých priorit pro různé operace, jelikož některé procesy výměny dat mezi přístroji a Service nejsou tak důležité jako jiné. Například procesy, se kterými klient nepracuje přímo v reálném čase jako je stahování archivů z přístroje, není nutné jejich požadavek vykonat okamžitě, pokud se ve stejnou dobu snaží spojit s přístrojem proces, na jehož odpovědi činnost klienta závisí. Důležitým procesem může být identifikace přístroje nebo stahování aktuálních naměřených dat a několik dalších.

Pro řešení priorit jednotlivých činností jsem zavedl opět globální pole proměnných nyní však typu *integer*. V programu proměnná obdržela název *hi_priority*. Používám zatím pouze dva stupně důležitosti operací. Vzniklé skupiny se dají nazvat jako skupina běžných činností a skupina důležitých činností. Skupina důležitých je při komunikaci s měřícími přístroji upřednostněna na úkor běžných činností. K nastavení priority pro jednotlivé operace dochází po zavolání funkce příslušející k obdržené zprávě. Před spuštěním operací sloužících ke komunikaci s přístrojem dojde k inkrementování hodnoty proměnné *hi_priority* (viz Výpis 4.3.2) s indexem příslušejícím k danému přístroji, obdobně jako u *Link_available*. Rozdílem oproti algoritmu s *Link_available* je použití typu *integer* oproti typu *boolean*. *Integer* jsem použil s ohledem na to, že zpráv s vyšší důležitostí může Service obdržet libovolné množství, aniž by je stihl zpracovat. Tímto způsobem se do proměnné ukládá aktuální počet upřednostněných akcí čekajících na zpracování. Při každém dokončení operace s vyšší důležitostí dojde k dekrementování hodnoty proměnné *hi_priority*. Ve chvíli, kdy jsou všechny důležité činnosti spolupracující s měřícím přístrojem dokončeny, hodnota *hi_priority* dosáhne opět své počáteční nulové hodnoty. Před spuštěním komunikace s přístrojem z procesu s běžnou prioritou dochází ke kontrole vyšší priority. Kontrola je prováděna pomocí podmínek *while* nebo *if* (viz Výpis 4.3.3) obdobně jako při ověřování dostupnosti linky. Pro kontrolu proměnných *Link_available* a *hi_priority* stačila vždy pouze jedna podmínka v kombinaci s logickou operací OR nebo AND.

```
hi_priority[Ms.hd.Device]++;
Ms.bd.SmpIdentify = (SmpIdentify)CommunicateSMP(Mes, typeof(SmpIdentify));
hi_priority[Ms.hd.Device]--;
```

Výpis 4.3.2: Ukázka nastavení priority

```
while (!link_available[Mes.hd.Device] || hi_priority[Mes.hd.Device] != 0)
        {
           System.Threading.Thread.Sleep(wait);
        }
while (link_available[Mes.hd.Device] && hi_priority[Mes.hd.Device] == 0)
        {
              …
        }
```

Výpis 4.3.3: Kontrola priority

4.4 Stahování archivů

Princip ukládání naměřených hodnot v přístroji

Měřící přístroj ukládá po předem nastavených intervalech naměřené hodnoty do své paměti. Každý blok uložených hodnot má v paměti přidělenou adresu podle které se dá zpětně vyhledat. Číslování adres v paměti přístroje začíná od 0 a končí zaplněním celé paměti nebo nastavenou hodnotou. Po té, co je uložen záznam s poslední adresou v řadě, začnou se záznamy ukládat opět od 0 a přepisovat tak staré záznamy.

Ve chvíli, kdy Service obdrží od klienta zprávu s požadavkem „GETARCHIVE", spustí se funkce, vytvořená k zajištění stažení a uložení žádaných archivů. Před začátkem stahování se přečtou hodnoty ze struktury *ArcSetting* přijaté v těle zprávy od klienta. Pro přenášení informací potřebných ke stahování archivů jsem vytvořil třídu *ArcSetting* (viz Výpis 4.4.1). Třída obsahuje pět proměnných. Proměnná určující typ požadovaného archivu je typu *byte* a má název *ArcType*. Interval záznamů ke stažení je dán dvěma proměnnými typu *uint* s označením *ArcStartAdress* pro začátek a *ArcStopAdress* pro konec intervalu. Archivy jsou z přístrojů stahovány po blocích, jejichž velikost je určena součinem proměnných *ArcCount* a *ReplyCount.*

```
public class ArcSetting
{
    public byte ArcType
    { get; set; }
    public uint ArcStartAddress
    { get; set; }
    public uint ArcStopAddress
    { get; set; }
    public ushort ArcCount
    { get; set; }
    public byte ReplyCount
    { get; set; }
}
```

Výpis 4.4.1: Třída s parametry pro stahování archivů

Před začátkem stahování archivních záznamů Service stáhne z přístroje několik typů konfiguračních nastavení a dále vytvoří SQL databázi, do které se budou stažené archivy ukládat. Pro stažení všech potřebných nastavení je spuštěn *for* cyklus (viz Výpis 4.4.2), ve kterém dojde ke stažení archivů v požadovaném rozsahu po zadaných krocích a současně uloží stažená data do databáze. K přístupu do databáze je použito *Xpo* a pro bezproblémové ukládání dat je uvolněna po každých 50 cyklech (stažených blocích) jeho cache. XPO je nástroj k ORM (objektově-relační mapování) z knihoven DXperience společnosti Developer Express. Účelem XPO je spojení objektově orientovaného jazyka (C#) s databází. Zajišťuje transformaci objektu do podoby vhodné k uložení do vybrané databáze. Uložená podoba musí být taková, aby bylo možné objekt z databáze opět ve stejné podobě načíst. Chyby, které nastanou během procesu stahování jsou ukládány do proměnné typu *DownloadProgresSummary* z knihoven KMB-lib. Mezi problémy, které jsou během procesu stahování zaznamenávány, patří kontrola *CRC* součtu každého přijatého bloku dat, dále je kontrolována správnost odpovědi, zda zpráva obsahuje požadovaná data, pokud Service neobdrží odpověď ve vymezeném čase, uloží se záznam o vypršení časového limitu před přijetím zprávy. Poslední zaznamenávaným problémem je kontrola uložení stažených dat do databáze.

Výše popsaný způsob pracuje spolehlivě v případě stahování jednoho archivu dat z jednoho přístroje. Při stahování více typů archivů najednou z jednoho přístroje by nedocházelo k jejich pravidelnému střídání, ale stahoval by se vždy archiv, který si linku k přístroji zarezervoval dříve.

```
for (uint ArcAddress = Mes.bd.ArcSetting.ArcStartAddress;
                ArcAddress < Mes.bd.ArcSetting.ArcStopAddress;
                ArcAddress += step)
```

Výpis 4.4.2: Cyklus pro stahování archivů

Popsaný proces stahování více archivů by byl neuspořádaný a nebylo by jisté, v jakém pořadí se jednotlivé požadavky dokončí. Musel jsem tedy vymyslet algoritmus, který by byl schopen zajistit střídavé stahování jednotlivých archivů.

Jelikož je aplikace Service multi-procesová, pro řízení stahování více archivů jsem zavedl opět globální proměnné. Počet spuštěných procesů stahujících archivní záznamy je uložen v proměnné *NumberDownArch* typu *integer*. Jak jsem již zmínil, archiv je stahován z přístroje po částech. Při stahování více archivů dostane přístup k přístroji další proces po dokončení aktuálně stahované části archivu prvního procesu. Počítadlo, které ukládá počet procesů, které již získaly přístup k přístroji v daném cyklu nese název *NumberDownedArch*. Proměnná *ArcSwitch* je deklarována jako pole typu integer. Slouží jako přepínač mezi jednotlivými archivy. Nastavuje se pro každý index v rozsahu 0-2. Poslední globální proměnnou pro stahování archivů je *ArcPoint*, která slouží jako ukazatel ukazující na jednotlivé položky pole *ArcSwitch* podle archivu, který se stahuje. Ve funkci pro stahování archivů je lokální proměnná *InnerArchNumber*, v této proměnné je uloženo identifikační číslo pro každý stahovaný archiv. Číslo je přiděleno podle hodnoty z pole *ArcSwitch*. Index první nalezené položky v poli s hodnotou 0 je přidělen jako ID číslo daného procesu stahování archivu.

Jak již bylo zmíněno, proměnná *ArcSwitch* může obsahovat tři hodnoty, pro každou položku v poli. Každá položka, která má hodnotu 0 značí, že její index nebyl přidělen žádnému stahovanému archivu a může tak být příště obsazena. Pokud nabývá hodnoty 2, značí to, že archiv s daným indexem existuje a zároveň je první na řadě při stahování. Hodnota 1 opět značí existenci procesu stahující archiv s daným ID, ale jeho stahování nezačne dříve než se změní hodnota na 2.

Po přidělení ID čísla dojde ke zjištění počtu stahovaných archivů (viz Výpis 4.4.3), v případě stahování pouze jednoho archivu je hodnota *ArcSwitch* okamžitě nastavena na 2, což značí, že může okamžitě začít stahování daného archivu. V opačném případě, kdy je archivů stahováno více nastaví se hodnota *ArcSwitch* na 1, spustí se cyklus *for* (viz Výpis 4.4.4) a proces se následně před začátkem stahování zacyklí v čekací smyčce do doby než mu je přidělena hodnota 2. Po každém stažení bloku archivu (jeden cykl *for*) dojde k navýšení hodnoty ukazatel *ArcPoint* a počítadla stažených archivů v jednom cyklu *NumberDownedArch* o 1. Dále dojde k porovnání proměnných *NumberDownedArch* s proměnou obsahující počet stahovaných archivů *NumberDownArch*. Při porovnávání se určí zda byl z každého stahovaného archivu stažena nastavená část, pokud ano dojde k nastavení ukazatele na 1 a začne tedy ukazovat opět na první archiv v pořadí. Dalším ověřením je, zda ukazatel neukazuje v poli *ArcSwitch* na hodnotu 0, případně dojde k upravení hodnoty *ArcPoint* na hodnotu, která odpovídá ID číslu následujícího archivu. Po upravení hodnot dojde k nastavení hodnoty v *ArcSwitch* na 2, diky které se může archiv s ID číslem odpovídajícím hodnotě *ArcPoint* začít stahovat. Na konci celé smyčky se provádí ověření zda už daný archiv není stažen celý. V pozitivním případě se na jeho místě v *ArcSwitch* nastaví 0, značící volnou pozici a dále dojde k snížení počtu stahovaných archivů o 1 v proměnné *NumberDownArch*.

```
If (NumberDownArch == 1)
{
   ArcSwitch[NumberDownArch] = 2;
   ArcPoint = 1;
}
else
{
   ArcSwitch[InnerArchNumber] = 1;
}
```

Výpis 4.4.3: Ověření počtu stahovaných archivů

```
for (uint ArcAddress = …..... ArcAddress += step)        {
            while (ArcSwitch[InnerArchNumber] != 2)          {
                  System.Threading.Thread.Sleep(wait);
            }
            while (….&& ArcSwitch[InnerArchNumber] == 2)
            {       ----------------------
OPERACE STAHUJÍCÍ A UKLÁDAJÍCÍ JEDNOTLIVÉ ARCHIVY
                  ----------------------
               ArcPoint++;
               NumberDownedArch++;
               if (NumberDownedArch >= NumberDownArch)
               {
                  ArcPoint = 1;
                  NumberDownedArch = 0;
               }
               while (ArcSwitch[ArcPoint] == 0)
               {
                  ArcPoint++;
               }
               ArcSwitch[ArcPoint] = 2;
            }
            if (ArcAddress+step >= Mes.bd.ArcSetting.ArcStopAddress)
            {
               ArcSwitch[InnerArchNumber] = 0;
               NumberDownArch--;
            }
```

Výpis 4.4.4: Ukázka algoritmu pro střídavé stahování archivů

4.5 Seznam měřících přístrojů

Pro ulehčení práce s klientem, aby si uživatel nemusel pamatovat adresy všech dostupných přístrojů, Service umožňuje jejich uložení do seznamu. Správu seznamu má na starosti Service, jelikož klientů může být spuštěno více a z různých počítačů a uživatel by tak seznam přístrojů musel zadávat každé klientské aplikaci zvlášť. Navíc by se při každé komunikaci klienta se Service musely ve zprávě přenášet všechny informace potřebné ke spojení s požadovaným měřícím přístrojem.

Nejprve jsem musel rozhodnout, co vše je nutné do seznamu přístrojů ukládat. Základem byla proměnná typu *integer*, pomocí které jsou jednotlivé přístroje identifikovány při komunikaci s klientem. Jedná se o stejná identifikační čísla, která zasílá klient v hlavičce zprávy pod názvem *Mes.hd.Device*, tyto čísla identifikující jednotlivé přístroje klient obdržel právě na základě vytvořeného seznamu. Za pomoci identifikačního čísla Service musí navázat spojení s vybraným přístrojem. Seznam tedy musí obsahovat informace o umístění jednotlivých přístrojů, v případě komunikace za pomoci TCP je nutné uložit IP adresu a port, pokud se přístroj nachází na sériovém portu COM, je potřeba ukládat číslo portu, na kterém je přístroj připojen, ale dále také nastavení komunikace jako je rychlost, parita, stopbity apod. Aby klient mohl rozlišovat jednotlivé přístroje uložené v seznamu podle jejich typu, do seznamu se navíc ukládá struktura SmpIdentify, která obsahuje identifikační údaje jednotlivých přístrojů. Pro všechny položky, které je nutné do seznamu ukládat jsem vytvořil novou třídu s názvem *AvailableDevices* (viz Výpis 4.5.1). V třídě *AvailableDevices* jsem použil třídy *TCPSetting* pro přístroje používající TCP a *COMSetting* pro přístroje používající sériovou linku. Třída *TCPSetting* (viz Výpis 4.5.2) obsahuje dvě proměnné, první nese název *IP* typu *string*, do které se ukládá IP adresa a port přístroje. Druhou proměnnou je *DeviceAddress* typu *byte,* která udává adresu přístroje v daném „komunikačním kanálu". Jak jsem již zmiňoval, nastavení komunikace pro přístroj na sériové lince je složitější než v případě ethernetu. V třídě *COMSetting* (viz Výpis 4.5.3) jsem použil proměnné *Port* typu *string*, do které se ukládá číslo portu, ke kterému je přístroj přípojen, *Portspeed* obsahující rychlost komunikace, *Parity* typu *parity*, jedná se o typ, u kterého je možné nastavit pět druhů parity, a to *none*, *even*, *mark*, *odd* a *space*. Pro proměnou *StopBits* jsem použil typ *stopbits,* jehož nastavení je obdobné jako v případě typu *parity*, nyní je však možné nastavit proměnnou na hodnoty *none*, *one*, *onepointfive* a *two*. Poslední dvě proměnné jsou typu *integer* v případě *DataBits* a *byte* v případě *DeviceAddress*, která má stejný význam jako v třídě *TCPSetting*.

V Service jsem vytvořil *List* třídy *AvailableDevices* pomocí příkazu *List<AvailableDevices>*. Tím vznikl úložný prostor pro informace o přidaných měřících přístrojích. Po vytvoření seznamu jsem musel v Service vytvořit také funkce pro jeho správu jako je přidávání, odebíraní, hledání dostupných přístrojů a další.

```
public class AvailableDevices
{
    [XmlElement("DevIdentNumber")]
    public int DevIdentNmbr;
    [XmlElement("Identify")]
    public KMB.Structures.SMP.SmpIdentify Identify;
    [XmlElement("TCPSetting")]
    public TCPSetting TCPSett;
    [XmlElement("COMSetting")]
    public COMSetting COMSett;
    public AvailableDevices()
    { }
    public AvailableDevices(int DevIdentNumber, KMB.Structures.SMP.SmpIdentify
                            SmpIdentify, TCPSetting TCPSetting, COMSetting COMSetting)
    {
        DevIdentNmbr = DevIdentNumber;
        Identify = SmpIdentify;
        TCPSett = TCPSetting;
        COMSett = COMSetting;
    }
}
```

Výpis 4.5.1: Třída AvailableDevices

```
public class TCPSetting
{
    public string IP
    { get; set; }
    public byte DeviceAdress
    { get; set; }
}
```

Výpis 4.5.2: Třída TCPSetting

```
public class COMSetting
{
   public string Port
   { get; set; }
   public int PortSpeed
   { get; set; }
   public int DataBits
   { get; set; }
   public Parity Parity
   { get; set; }
   public StopBits StopBits
   { get; set; }
   public byte DeviceAdress
   { get; set; }
}
```

Výpis 4.5.3: Třída COMSetting

Hledání dostupných přístrojů

O tuto činnost se v Service stará funkce *GetAllDev*. Spolu s příkazem ke spuštění funkce pro hledání přístrojů přijde ve zprávě od klienta také *List* třídy *AvailableDevices* s IP adresami, na kterých má Service přístroje hledat a také nastavení komunikace COM portů pro vyhledávání přístrojů na sériové lince. Třída *AvailableDevices* sice primárně nebyla vytvořena pro tyto účely, ale svou funkci zde plní dostatečně. Vyhledávání přístrojů začne nejprve na COM portech počítače. Je zavolána funkce, která vyhledá všechny dostupné COM porty v počítači, uloží je do pole proměnných typu *string* a pole vrátí zpět funkci *GetAllDev*. Pro každý port je vytvořen *for* cyklus od 1 do 255, což je maximum možných přístrojů na jednom portu. S každým portem je navázáno spojení a odeslána zpráva pro identifikaci přístroje na adrese odpovídající indexu cyklu. Pokud Service ve stanoveném čase neobdrží odpověď, odešle dotaz přístroji na další adrese. V případě, že odpověď obdrží, přidělí přístroji identifikační číslo začínající od 1 pro první nalezený přístroj. Následně uloží příslušné údaje do seznamu přístrojů. Pokud je na jednom kanálu pouze jeden přístroj, odpovídá na všechny dotazy od 1 do 255, proto bylo nutné navíc porovnávat adresu přístroje, kterou se identifikoval s indexem cyklu, nerovnají-li se znamená to, že na lince je pouze jeden měřící přístroj. Obdobným

způsobem dochází k vyhledávání ethernetových přístrojů. Hlavním rozdílem je načítání IP adres, na kterých by se měly přístroje nacházet, z *Listu* obdrženého od klientské aplikace. Po identifikování všech měřících přístrojů a jejich uložení do seznamu je celý seznam odeslán klientské aplikaci. Po odeslání seznamu klientovi je zavolána funkce pro uložení seznamu do souboru.

Uložení seznamu přístrojů

Pro tuto činnost jsem v Service vytvořil funkci *SaveDevToFile* (viz Výpis 4.5.4). Seznam je uložen do xml souboru s použitím *streamwriteru.* Vzhled seznamu přístrojů uloženého v souboru je velice podobný formátu zpráv přenášených mezi klienty a Service, jelikož jsem se rozhodl pro usnadnění použít stejný protokol. Do souboru se pomocí serializace z protokolu ukládá pouze tělo zprávy *MsgBody*. Uložení pouze těla zprávy bez hlavičky je docíleno přidělením hodnoty *null* hlavičce zprávy.

```
public static void SaveDevToFile()
    {
        Msg Mes = new Msg();
        try
        {
            StreamWriter wr = new StreamWriter("DevList.xml");
            Mes.hd = null;
            Mes.bd.ListAvailableDevices = AvDevL;
            x.Serialize(wr, Mes);
            wr.Flush();
            wr.Close();
        }
        catch
        {
                ...
        }
    }
```

Výpis 4.5.4: Ukládání seznamu přístrojů

Odebírání přístrojů ze seznamu

Odebírání přístrojů je zajištěno funkcí *RemoveDev* (viz Výpis 4.5.5). K odebrání položky je použita metoda *RemoveAt*, která má za parametr index, který značí umístění přístroje v seznamu. Jelikož poloha jednotlivých přístrojů v seznamu se mění v závislosti na jejich dodatečném přidávání a odebírání, nebylo možné přidělovat přístrojům identifikační čísla podle umístění v seznamu. Proto je pro vyhledání indexu přístroje v seznamu, který klient chce odebrat, použita metoda *FindIndex*. Metoda *FindIndex* vyhledá zadanou položku v seznamu a vrátí její index. V mém případě, kdy je klientské aplikaci známo pouze identifikační číslo přístroje, je hledání založeno na porovnávání identifikačních čísel v seznamu s číslem obdrženým od klienta. Po nalezení odpovídajícího indexu je položka ze seznamu odstraněna, aktualizovaný seznam je opět uložen do souboru a následně také odeslán klientovy.

```
public static void RemoveDev(Msg Mes)
    {
        Msg Ms = new Msg();
        try
        {
  AvDevL.RemoveAt(AvDevL.FindIndex(ind => ind.DevIdentNmbr == Mes.hd.Device));
            Ms.bd.ListAvailableDevices = AvDevL;
        }
        catch
        {
            Ms.bd.ServiceMessage = "Error while removing device, device not found";
            ...
            return;
        }
        Ms.hd = Mes.hd;
        SaveDevToFile();
        SerializeAndSend(Ms);
    }
```

Výpis 4.5.5: Odebírání přístrojů

Přidávání přístrojů

Funkci pro přidávání měřících přístrojů do seznamu jsem pojmenoval *AddDev*. Klient přidává přístroj pomocí IP adresy a adresy přístroje v případě ethernetu nebo čísla COM portu a jeho nastavení spolu s adresou přístroje v případě přístroje na sériové lince. Funkce *AddDev* v Service nejprve určí, zda se přidává ethernetový přístroj nebo přístroj přes sériovou linku. Následně je s přístrojem navázáno spojení, v případě, že je spojení navázáno úspěšně, je přístroji zaslán dotaz na identifikaci. V jiném případě je klient informován o problému s hledaným přístrojem. Adresy přístrojů, již přidaných do seznamu se porovnají s adresou aktuálně identifikovaného přístroje, aby nedošlo k duplikování přístroje. V opačném případě je přístroj přidán do seznamu. Pro přidělení identifikačního čísla přístroje jsem vytvořil metodu *GetNextDevID* (viz Výpis 4.5.6). Funkce obsahuje *for* cyklus, který prohledá seznam přístrojů a najde případné chybějící identifikační číslo v posloupnosti již přidělených čísel. Mezera v identifikačních číslech může vzniknout například při odstranění některého z přístrojů ze seznamu. Číslo je po té přiděleno aktuálně přidanému přístroji. Pokud žádná mezera v posloupnosti není, nový přístroj jednoduše obdrží následující číslo.

```
public static int GetNextDevID()
{
   int ind = 0;
    AvDevL.OrderBy(index => index.DevIdentNmbr);
   for (ind = 0; ind < AvDevL.Count; ind++)
   {
      if (AvDevL[ind].DevIdentNmbr != ind + 1)
      {
         break;
      }
   }
   return ind;
}
```

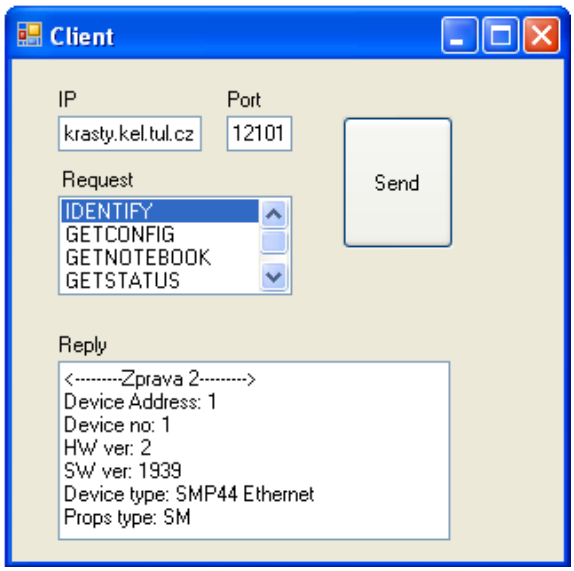
Výpis 4.5.6: Funkce pro přidělení ID čísla přístroji

5 Dosažené výsledky

5.1 Testovací aplikace

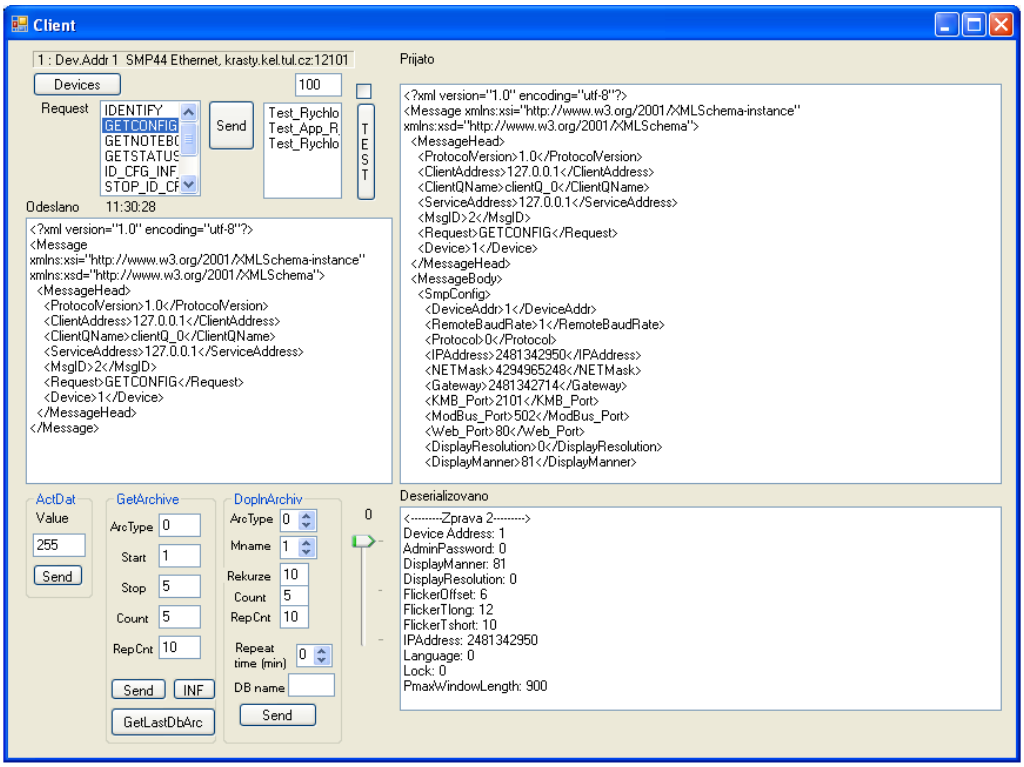
Za účelem ověřování vytvořených funkcí a testování správné funkčnosti celé aplikace jsem souběžně s tvorbou Service vytvářel i klientskou aplikaci. Vývoj probíhal postupně v závislosti na potřebách Service. Základním kamenem je část programu zjišťující komunikaci pomocí MSMQ, jejíž podoba je téměř shodná s funkcí pro zajištění MSMQ v Service.

První verze aplikace byla schopná komunikovat najednou pouze s jedním přístrojem. Adresa přístroje se zadávala do textboxu a při změně přístroje musela být přepsána na novou. V *listboxu* byly uloženy formáty jednotlivých dotazů, na které bylo možné se u Service dotázat. Postupným rozšiřováním schopností Service jsem rozšiřoval i seznam dotazů v *listboxu*. Typy dotazů byly omezeny pouze na dotazy bez doplňujících parametrů. Jednalo se hlavně o dotazy na identifikaci a nastavení přístroje. Přijaté odpovědi se zobrazovaly v textovém poli, v podobě jakou mají po svém uložení do struktury.



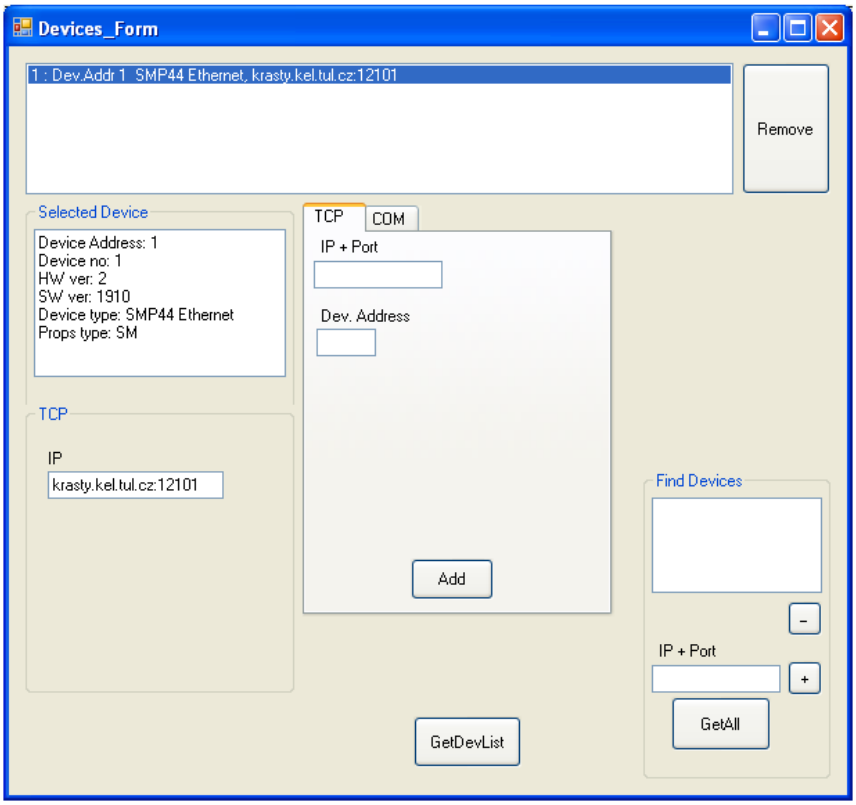
Obr. 5.1: První verze Testovací aplikace

S postupným vývojem jsem došel k závěru, že je nutné zobrazovat i XML tvar odesílaných a přijímaných zpráv v podobě, v jaké jsou přenášeny frontou zpráv. Toto zobrazení bylo nutné hlavně pro kontrolu podoby zpráv při doplňování nových struktur a funkcí, jelikož každý dotaz obsahující doplňující parametry má odlišný tvar. Nebylo možné je všechny sjednotit obdobně jako ty bez parametru. Rozhodl jsem se tedy použít jen ty nejdůležitější. Nejprve jsem otestoval přenos parametrů na jednodušším typu dotazu, kterým byla žádost o aktuální data, protože dotaz obsahuje pouze jeden parametr. Po důkladném vyzkoušení jsem do aplikace přidal požadavek na stažení archivních záznamů i s potřebnými parametry.



Obr. 5.2: Současná podoba testovací aplikace

Téměř finální podobu testovací aplikace získala po přidání podpory pro seznam přístrojů obsažený v Service. Aplikaci jsem doplnil o další formulář, ve kterém je možné testovat funkce spojené s jednotlivými přístroji uloženými v Service. Přístroje je možné s pomocí tohoto formuláře přidávat a mazat ze seznamu, případně konfigurovat způsob komunikace s nimi. Dále je zde *listbox,* zobrazující všechny dostupné přístroje v seznamu uloženém v Service, který zároveň umožňuje vybrat přístroj, se kterým se bude komunikovat.



Obr. 5.3: Formulář pro správu seznamu přístrojů

5.2 Protokol zprávy

Funguje jako samostatná dll knihovna. Její funkce využívá jak testovací aplikace tak Service. Hlavním účelem knihovny je správné formátování odesílaných zpráv a následně také zajišťuje jejich správné přečtení po přijetí. Vedlejší funkcí protokolu je správa doplňujících struktur, které nebyly obsaženy v knihovnách KMB.

Během vývoje aplikací se podoba protokolu měnila hlavně v podobě podporovaných struktur přenášených v těle zprávy. Hlavičku zprávy bylo nutné v průběhu vývoje výrazně změnit pouze jednou. Tato změna proběhla v době vytvoření funkcí pro ukládání seznamu přístrojů v Service. Nyní již není nutné přenášet v hlavičce zprávy veškeré informace nutné pro spojení s přístrojem. Po změně se přenáší pouze identifikační číslo přístroje na jehož základě Service rozezná, se kterým přístrojem má navázat spojení.

5.3 Multi-threading

Pro každou zprávu, kterou Service přijme, je vytvořeno nové vlákno, ve kterém je činnost, kterou zpráva požaduje vykonána. Aplikace je tak neustále dostupná všem klientům a schopná přijímat jejich dotazy. Všechny činnosti jsou zpracovávány najednou, neboli paralelně, do doby než se budou chtít dvě nebo více různých vláken spojit s jedním měřícím přístrojem, jelikož přístroj může zpracovávat pouze jeden příkaz v jeden okamžik. Ve chvíli, kdy nastane tato situace je upřednostněno vlákno, které vykonává činnost s větší prioritou nebo v jiných případech vlákno, které požádalo o spojení s přístrojem dříve.

5.4 Integrace do stávajícího vizualizačního programu

Service je v současné době s vizualizační aplikací integrován na úrovni společné databáze. Obě aplikace jsou schopné do společné databáze data ukládat, případně je zní i podle požadavků a potřeb načítat. Hlubší provázání obou aplikací bude s největší pravděpodobností realizováno v rámci některého z dalších projektů.

5.5 Stahování archivních záznamů

Jedna z hlavních funkcí Service. V současné době existuje sedm druhů archivů, které přístroj ukládá. Service je schopný, na základě požadavků klienta, z přístroje stáhnout jakýkoliv z uvedených záznamů. Vybraný interval uložených záznamů je stahován po klientem určených částech a stažená data jsou postupně ukládána do databáze určené před začátkem stahování. Při stahování jsou ukládány informace o chybách jako je chybná odpověď přístroje, CRC neodpovídající obsahu zprávy, vypršení času vymezeného pro spojení s přístrojem a nakonec dochází také k ukládání chyb vzniklých při zápisu dat do databáze. Po dokončení stahování celého intervalu archivu jsou informace o dokončení činnosti a o vzniklých chybách odeslány klientovi.

V ukázce průběhu stahování archivu (viz Výpis 5.1) měl Service za úkol stáhnout a uložit do databáze archivní záznamy v intervalu 2850-3025. Z požadavků klienta na hodnoty ArcCount a ReplyCount, byla vypočítána velikost kroku na 50. Typem archivu je 0, což odpovídá záznamům *MainArchive*. Po načtení všech proměnných došlo k připojení do požadované databáze a následně k uvolnění cache XPO, ke kterému dochází vždy při prvním a následně každém padesátém intervalu stahování. V prvním běhu, v rozsahu 2850-2900, je vidět problém s uložením prvních tří

záznamů do databáze. Tento problém nejčastěji signalizuje, že data byla na uvedené pozice v databázi uložena již dříve a muselo by tedy dojít k jejich přepsání. Z uvedených časů je možné vyčíst, že uložení 50 záznamů do databáze trvá podstatně déle než jejich stažení z ethernetového přístroje. Rychlost stahování dat z přístroje je do určité míry ovlivněna i kvalitou spojení mezi přístrojem a Service, ale v mém případě bylo ukládání do databáze 8-10x pomalejší, než samotné stažení, jak je vidět i v další ukázce (viz Výpis5.2), tentokrát však s větším intervalem záznamů. Po několika naměřených časech a různých délkách intervalu je zřejmé, že uložení jednoho záznamu do databáze trvá přibližně 45 ms.

```
[15.5.2009 12:52:33]  Rozsah: 2850 – 3025, ArcCount, ReplyCount: 5, 10,   Typ archivu: 0
[15.5.2009 12:52:36]  Pripojeno k DB
[15.5.2009 12:52:37]  Zacatek stahovani intervalu :  2850-2900
[15.5.2009 12:52:37]  Cache XPO uvolnena
[15.5.2009 12:52:37]  Odeslani zadosti o interval :  2850-2900 pristroji
[15.5.2009 12:52:37]  Interval archivu   2850-2900 prijat v case: 295 ms
[15.5.2009 12:52:37]  Ukladani stazeneho intervalu do DB
[15.5.2009 12:52:37]  Chyba pri ukladani do DB v cyklu  0
[15.5.2009 12:52:38]  Chyba pri ukladani do DB v cyklu  1
[15.5.2009 12:52:38]  Chyba pri ukladani do DB v cyklu  2
[15.5.2009 12:52:40]  Stazeny interval ulozen do DB v  50 cyklech za 2329 ms
[15.5.2009 12:52:40]  Interval 2850-2900 dokoncen v case: 2918 ms
[15.5.2009 12:52:40]  ----------------------------------------------------
[15.5.2009 12:52:40]  Zacatek stahovani intervalu :  2900-2950
[15.5.2009 12:52:40]  Odeslani zadosti o interval :  2900-2950 pristroji
[15.5.2009 12:52:40]  Interval archivu   2900-2950 prijat v case: 402 ms
[15.5.2009 12:52:40]  Ukladani stazeneho intervalu do DB
[15.5.2009 12:52:42]  Stazeny interval ulozen do DB v  50 cyklech za 1906 ms
[15.5.2009 12:52:42]  Interval 2900-2950 dokoncen v case: 2332 ms
[15.5.2009 12:52:42]  ----------------------------------------------------
[15.5.2009 12:52:42]  Zacatek stahovani intervalu :  2950-3000
[15.5.2009 12:52:42]  Odeslani zadosti o interval :  2950-3000 pristroji
[15.5.2009 12:52:42]  Interval archivu   2950-3000 prijat v case: 1295 ms
[15.5.2009 12:52:42]  Stazeny interval archivu ma hodnotu null
[15.5.2009 12:52:44]  Interval 2950-3000 dokoncen v case: 1484ms
[15.5.2009 12:52:44]  ----------------------------------------------------
[15.5.2009 12:52:44]  Zacatek stahovani intervalu :  3000-3050
```

```
[15.5.2009 12:52:44]  Odeslani zadosti o interval :  3000-3025 pristroji
[15.5.2009 12:52:44]  Interval archivu   3000-3025 prijat v case: 169 ms
[15.5.2009 12:52:44]  Ukladani stazeneho intervalu do DB
[15.5.2009 12:52:45]  Stazeny interval ulozen do DB v  25 cyklech za 960 ms
[15.5.2009 12:52:45]  Interval 3000-3025 dokoncen v case: 1152 ms
[15.5.2009 12:52:45]  ----------------------------------------------------
[15.5.2009 12:52:45]  STAHOVANI ARCHIVU V ROZSAHU 2850 - 3025 DOKONCENO
[15.5.2009 12:52:45]  Intervaly se spatnym CRC: 0
[15.5.2009 12:52:45]  Spatne odpovedi: 1
[15.5.2009 12:52:45]  Timeout: 0
[15.5.2009 12:52:45]  Chyby ulozeni do DB: 1
```

Výpis 5.1: Průběh stahování archivu

```
[15.5.2009 14:24:08]  Odeslani zadosti o interval :  3500-4000 pristroji
[15.5.2009 14:24:10]  Interval archivu   3500-4000 prijat v case: 2233 ms
[15.5.2009 14:24:10]  Ukladani stazeneho intervalu do DB
[15.5.2009 14:24:29]  Stazeny interval ulozen do DB v  500 cyklech za 19483 ms
```

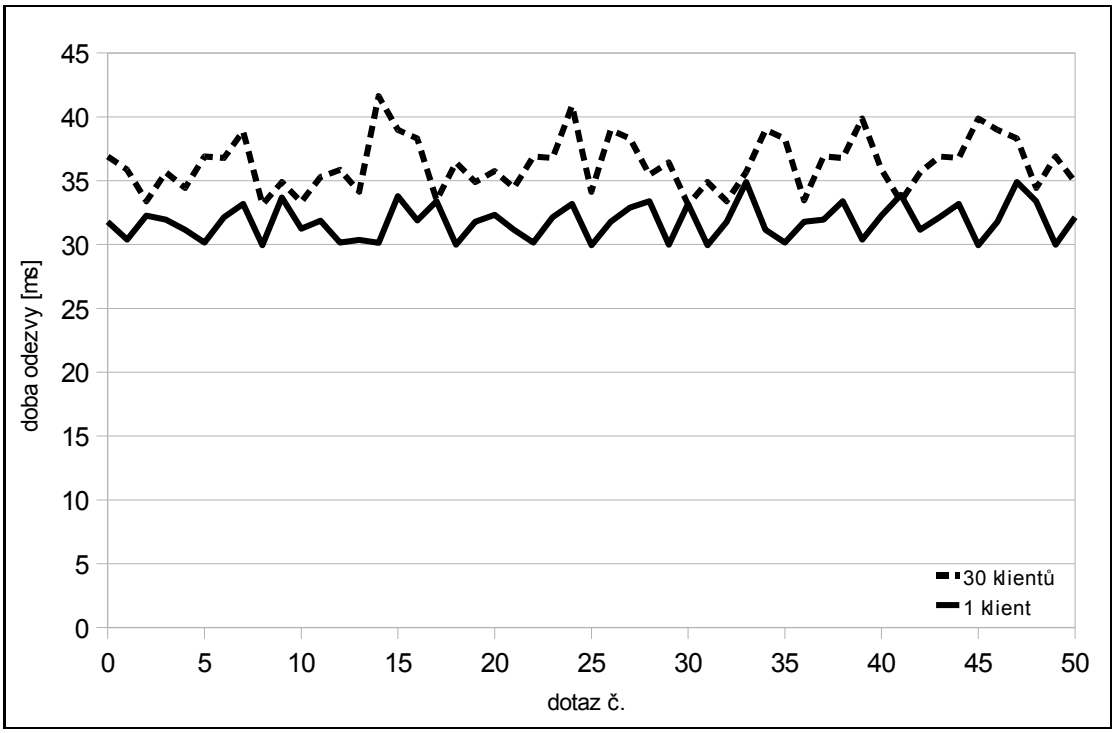
Výpis 5.2: Stažení většího intervalu záznamů

Další možností stahování archivních záznamů je jeho postupné doplňování do databáze. Při použití tohoto způsobu není potřeba zadávat stahovaný interval, ale pouze typ archivu a velikost kroku při stahování. Po příjmu zprávy se Service přípojí k předem zvolené databázi a vyhledá datum a čas obsažený v posledním uloženém záznamu. Po načtení tohoto času se spojí s vybraným přístrojem, kde vyhledá adresu záznamu odpovídající hodnotě v databázi. Na základě zjištěné adresy a adresy posledního záznamu v přístroji se vytvoří interval pro záznamy, které mají být do databáze doplněny. Další průběh stahovaní je již stejný jako v případě klasického způsobu stahování záznamů. Funkce pro postupné doplňování pracuje ve smyčce se zvoleným časovým zpožděním.

5.6 Rychlost aplikace

Při zjišťování rychlosti jsem se zaměřil na rychlost zpracování dotazu a také na schopnost zpracovat více dotazů od různých klientů. Dobu zpracování jsem měřil pomocí testovací aplikace. Změřený časový interval odpovídá době od odeslání dotazu po obdržení odpovědi. Jako dotaz jsem zvolil požadavek na zaslání seznamu dostupných přístrojů, patřící k jednomu z nejjednodušších typů dotazu. Při komunikaci

Service s jedním klientem se pohybovala průměrná doba odpovědi mezi 30 a 35 ms (viz Graf 1). Pokud by komunikace probíhala i s měřícím přístrojem, čas odpovědi by závisel ve větší míře na rychlosti konkrétního přístroje a kvalitě spojení mezi Service a přístrojem. Pro napodobení komunikace s větším množstvím klientů jsem spojil se Service více testovacích klientských aplikací najednou. Jako vhodný počet spojení jsem zvolil 30, což je podle mého názoru již dostatečný počet na prověření schopností Service. V tomto případě se měřené doba odezvy pro jednotlivé aplikace pohybovala v průměru okolo 35 maximálně mírně nad 40 ms (viz Graf 1). Tyto hodnoty v porovnání s časy při použití jedné aplikace považuji za velice slušné. Podstatně horších výsledků by bylo dosaženo při použití stejného počtu aplikací v kombinaci s komunikací s jedním přístrojem. Jelikož přístroj může zpracovávat požadavky pouze postupně, ostatní požadavky musejí čekat na uvolnění přístroje.



Graf 1: Doba zpoždění

5.7 Zjištěné nedostatky

Během testování vytvořených funkcí pro stahování a následné ukládání archivních záznamů jsem narazil na problém s přístupem aplikace k databázi. Problém spočívá v tom, že současná verze knihoven, které používám pro připojení a přístup do databáze neumožňuje připojit se k více databázím najednou. Pokud se připojí v jednom ze spuštěných vláken aplikace k databázi, toto připojení následně platí pro všechna vlákna globálně. Ukládání data do jednotlivých databází tedy musí být realizováno postupně, což je velice neefektivní.

5.8 Možné rozšíření aplikace

Dostupné Struktury

Jedním z hlavních rozšíření, které je nejen možné, ale s budoucím vývojem aplikace bude téměř jistě i uskutečňováno, jsou struktury přenášené ve zprávách. V závislosti na dalších požadavcích na aplikace používající knihovny protokolu zprávy je možné do těchto knihoven doplňovat dle potřeby nové struktury. Již při návrhu protokolu jsem s touto možností počítal. Není tak vůbec složité nové struktury doplnit dle potřeby.

Vytvoření systémové služby

V současné době je Service v podobě konzolové aplikace. Tato podoba je výhodná a dostačující pro testování funkcí a další vývoj aplikace. V budoucnu, než bude aplikace za nějakým účelem nasazena v praxi, by bylo výhodnější předělat jí na systémovou službu. Jako systémová služba by aplikace Service byla spuštěna na pozadí, kde by zpracovávala příkazy od klientů. Uživatel by s ní tak nepřišel žádným způsobem do styku a jeho zásahy by byly realizovány přes grafické prostředí, neboli klientskou aplikaci.

„Záplatování” archivních záznamů

V případě, že při stahování záznamů dojde k chybě, je chyba zaznamenána do logu, případně je o jejím vzniku informován klient. Po zaznamenání se však pokračuje ve stahování dalších částí záznamu a jejich ukládání do databáze. Chybné archivy již nejsou nahrazeny a v databázi tak vznikají „prázdná” místa bez hodnot. Tento problém by bylo možné vyřešit „záplatováním”. Jednalo by se o funkci, která by prohledala záznamy v databázi a našla vzniklá „prázdná” místa. Následně by se pokusila z přístroje chybějící záznamy stáhnout a uložit do databáze.

Ukládání archivních záznamů

Ukládání záznamů by bylo možné rozšířit o více způsobů jejich uložení. V současnosti nemá uživatel na výběr, jakým způsobem stažené záznamy uložit a jsou všechny uloženy do zvolené databáze. Vhodným způsobem ukládání, které by bylo možné realizovat, je ukládání dat do souboru. Tímto řešením by bylo snadné stažená data přenášet, případně zálohovat. Uložení do souboru je sice možné i při použití databáze, ale uživatel je nucen použít speciální program pro správu databází, což na něj klade další nároky.

6 Závěr

Jelikož jsem před touto prací nikdy nenavrhoval žádnou složitější aplikaci a zároveň jsem byl začátečník ohledně programování v jazyce C#, vstupoval jsem do „neznámých vod". Před započetím prací jsem se tedy nejdříve seznámil s jazykem C#, jelikož jsem se do té doby setkal pouze s podobným jazykem C++. Zmíněné C++ jsem studoval jeden semestr v předmětu zabývajícím se programováním, nebyl jsem tedy velký odborník ani v tomto jazyce. Získal jsem však alespoň základní znalosti. Při seznamování s C# bylo od počátku zřejmé, že se jedná o jazyk velice podobný C++ a podle mého názoru v určitých ohledech i jednodušší. Při návrhu aplikace jsem se naštěstí vyvaroval různých „slepých uliček" a postupoval jsem téměř přímo od prvních základů až po současnou verzi. Tvorba tohoto bakalářského projektu mi tedy přinesla hodně znalostí ohledně programování v .NET konkrétněji v prostředí C#. Dále jsem také získal zkušenosti s návrhem nové aplikace a komunikačního protokolu přímo pro ni.

Současná podoba aplikace plní zadané požadavky, ale zdaleka to není její finální podoba. Na jejím vývoji budu pracovat i po odevzdání bakalářské práce. Podoba, v jaké se nyní nachází tvoří jakousi základní kostru pro její další rozvoj. Tato kostra by se již neměla v budoucnu výrazně měnit a další rozšiřování bude spočívat hlavně v doplňování nových funkcí dle nejrůznějších požadavků a potřeby. Obdobným způsobem bude rozšiřována dle potřeb i knihovna protokolu zprávy. Vzhled Service pravděpodobně bude změněn do podoby, kdy nebude mít žádné grafické prostředí a jeho vstupem a výstupem budou pouze fronty zpráv, případně spojení s přístroji. Zatím nejpravděpodobněji se jeví jeho transformace do systémové služby. V podobě služby se následně nainstaluje do počítače a bude plnit svou funkci správy zpráv mezi klientskými aplikacemi a měřícími přístroji.

V budoucnu až Service dosáhne určité pokročilejší vývojové fáze bude podle nynějších plánů integrován do prostředí programu ENVIS společnosti KMB systems, s.r.o.. Zatím není jisté, kdy aplikace dosáhne finálního stádia, ale rád bych na vývoji Service dále pracoval a další poznatky a výsledky shrnul například v diplomové práci.

Seznam použité literatury

[1] *.NET Framework Developer's Guide* [online].

URL: ⟨http://msdn.microsoft.com/⟩.

[2] ALBAHARI, Joseph. *Threading in C#* [online].

URL: ⟨http://www.albahari.com/threading/⟩.

[3] *Dokumentace k měřícím přístrojům a programu RETIS* [online].

URL: ⟨http://www.kmb.cz/⟩.

[4] *Dokumentace ke knihovnám DXperience* [online].

URL: ⟨http://www.devexpress.com/⟩.

[5] GRUJIC, Dejan. *Introduction to MSMQ* [online].

URL: ⟨http://www.cogin.com/articles/IntroductionToMSMQ.php⟩.

[6] *Interprocess Communications* [online].

URL: ⟨http://msdn.microsoft.com/en-us/library/aa365574(VS.85).aspx⟩.

[7] *IPC (Interprocess Communication)* [online].

URL: ⟨http://www.linktionary.com/i/ipc.html⟩.

[8] JONES, Michael. *MSMQ for .NET Developers* [online].

URL: ⟨http://www.devx.com/dotnet/Article/27560/1954⟩.

[9] *Message Queuing (MSMQ)* [online].

URL: ⟨http://msdn.microsoft.com/en-us/library/ms711472(VS.85).aspx⟩.

[10] VIRIUS, Miroslav. *C# - Hotová řešení*, Computer Press, ISBN 8025110842

Příloha A – Formát zpráv zasílaných Service

Zprávy bez parametrů

Tělo MsgBody zprávy je u tohoto druhu zpráv prázdné (MsgBody=null). Formát jednotlivých zpráv je totožný(viz Výpis A.1), liší se pouze v požadavku (*Request)*.

```
<?xml version="1.0" encoding="utf-8"?>
<Message xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://
www.w3.org/2001/XMLSchema">
  <MessageHead>
    <ProtocolVersion>1.0</ProtocolVersion>
    <ClientAddress>127.0.0.1</ClientAddress>
    <ClientQName>clientQ_0</ClientQName>
    <ServiceAddress>127.0.0.1</ServiceAddress>
    <MsgID>0</MsgID>
    <Request>IDENTIFY</Request>
    <Device>1</Device>
  </MessageHead>
</Message>
```

Výpis A.1: Identifikace

Další dostupné požadavky pro zprávy bez parametrů

Odpověď vytvořena až v přístroji

<Request>GETCONFIG</Request> - stažení nastavení přístroje

<Request>GETNOTEBOOK</Request> - stažení informací o měření (název..)

<Request>GETSTATUS</Request> - stažení stavů jednotlivých měření (počet, časy..)

<Request>GETELMERTARIF</Request> - stažení tarifů elektroměru

Odpověď vytvořena v Service

<Request>REMOVEDEVICE</Request> - odstraní vybraný přístroj ze seznamu

<Request>GETDEVICE</Request> - žádost o seznam přístrojů

<Request>GETDBSQL</Request> - dotaz na dostupné SQL servery

Zprávy s parametry

Tělo zpráv obsahuje doplňující parametry k zadané činnosti.

```
<?xml version="1.0" encoding="utf-8"?>
<Message xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://
www.w3.org/2001/XMLSchema">
  <MessageHead>
    <ProtocolVersion>1.0</ProtocolVersion>
    <ClientAddress>127.0.0.1</ClientAddress>
    <ClientQName>clientQ_0</ClientQName>
    <ServiceAddress>127.0.0.1</ServiceAddress>
    <MsgID>2</MsgID>
    <Request>ACTDATA</Request>
    <Device>1</Device>
  </MessageHead>
  <MessageBody>
    <SmpMsgCfgInt>
      <Value>255</Value>
    </SmpMsgCfgInt>
  </MessageBody>
</Message>
```

– žádost o aktuální data z přístroje

```
<?xml version="1.0" encoding="utf-8"?>
<Message xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://
www.w3.org/2001/XMLSchema">
 <MessageHead>
  <ProtocolVersion>1.0</ProtocolVersion>
  <ClientAddress>127.0.0.1</ClientAddress>
  <ClientQName>clientQ_1</ClientQName>
  <ServiceAddress>127.0.0.1</ServiceAddress>
  <MsgID>4</MsgID>
  <Request>GETARCHIVE</Request>
  <Device>1</Device>
 </MessageHead>
 <MessageBody>
  <ArcSetting>
   <ArcType>6</ArcType>
   <ArcStartAddress>0</ArcStartAddress>
   <ArcStopAddress>2000</ArcStopAddress>
   <ArcCount>5</ArcCount>
   <ReplyCount>10</ReplyCount>
  </ArcSetting>
 </MessageBody>
</Message>
```

– příkaz pro stažení archivních záznamů daného typu v zadaném rozsahu

```
<?xml version="1.0" encoding="utf-8"?>
<Message xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://
www.w3.org/2001/XMLSchema">
 <MessageHead>
  <ProtocolVersion>1.0</ProtocolVersion>
  <ClientAddress>127.0.0.1</ClientAddress>
  <ClientQName>clientQ_1</ClientQName>
  <ServiceAddress>127.0.0.1</ServiceAddress>
  <MsgID>5</MsgID>
  <Request>ARCINFO</Request>
  <Device>1</Device>
 </MessageHead>
 <MessageBody>
  <ArcSetting>
   <ArcType>6</ArcType>
   <ArcStartAddress>0</ArcStartAddress>
   <ArcStopAddress>0</ArcStopAddress>
   <ArcCount>0</ArcCount>
   <ReplyCount>0</ReplyCount>
  </ArcSetting>
 </MessageBody>
</Message>
```

– žádost o rozsah daného typu archivních záznamů uložených v přístroji

```
<?xml version="1.0" encoding="utf-8"?>
<Message xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://
www.w3.org/2001/XMLSchema">
 <MessageHead>
  <ProtocolVersion>1.0</ProtocolVersion>
  <ClientAddress>127.0.0.1</ClientAddress>
  <ClientQName>clientQ_1</ClientQName>
  <ServiceAddress>127.0.0.1</ServiceAddress>
  <MsgID>8</MsgID>
  <Request>GETDBNAME</Request>
  <Device>1</Device>
 </MessageHead>
 <MessageBody>
  <DBSetting>
   <SQLServerName>
    <string>PC\SQLEXPRESS</string>
   </SQLServerName>
  </DBSetting>
 </MessageBody>
</Message>
```

– požadavek na seznam existujících databází v zadaném SQLServeru

```
<?xml version="1.0" encoding="utf-8"?>
<Message xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://
www.w3.org/2001/XMLSchema">
 <MessageHead>
  <ProtocolVersion>1.0</ProtocolVersion>
  <ClientAddress>127.0.0.1</ClientAddress>
  <ClientQName>clientQ_1</ClientQName>
  <ServiceAddress>127.0.0.1</ServiceAddress>
  <MsgID>9</MsgID>
  <Request>ARCLOOP</Request>
  <Device>1</Device>
 </MessageHead>
 <MessageBody>
  <ArcSetting>
   <ArcType>6</ArcType>
   <ArcStartAddress>0</ArcStartAddress>
   <ArcStopAddress>0</ArcStopAddress>
   <ArcCount>5</ArcCount>
   <ReplyCount>10</ReplyCount>
  </ArcSetting>
  <ArcLoopParam>
   <Recursion>5</Recursion>
   <RepeatTime>1</RepeatTime>
   <MeasName>1</MeasName>
  </ArcLoopParam>
  <DBSetting>
   <DBName>
    <string>ELMER</string>
   </DBName>
   <SQLServerName>
    <string>PC\SQLEXPRESS</string>
   </SQLServerName>
  </DBSetting>
 </MessageBody>
</Message>
```

– příkaz na postupné doplňování archivních záznamů do databáze

```
<?xml version="1.0" encoding="utf-8"?>
<Message xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://
www.w3.org/2001/XMLSchema">
 <MessageHead>
  <ProtocolVersion>1.0</ProtocolVersion>
  <ClientAddress>127.0.0.1</ClientAddress>
  <ClientQName>clientQ_1</ClientQName>
  <ServiceAddress>127.0.0.1</ServiceAddress>
  <MsgID>12</MsgID>
  <Request>ADDDEVICE</Request>
  <Device>1</Device>
 </MessageHead>
 <MessageBody>
  <AvailableDevices>
   <DevIdentNumber>0</DevIdentNumber>
   <TCPSetting>
    <IP>krasty.kel.tul.cz:12101</IP>
    <DeviceAdress>1</DeviceAdress>
   </TCPSetting>
  </AvailableDevices>
 </MessageBody>
</Message>
```

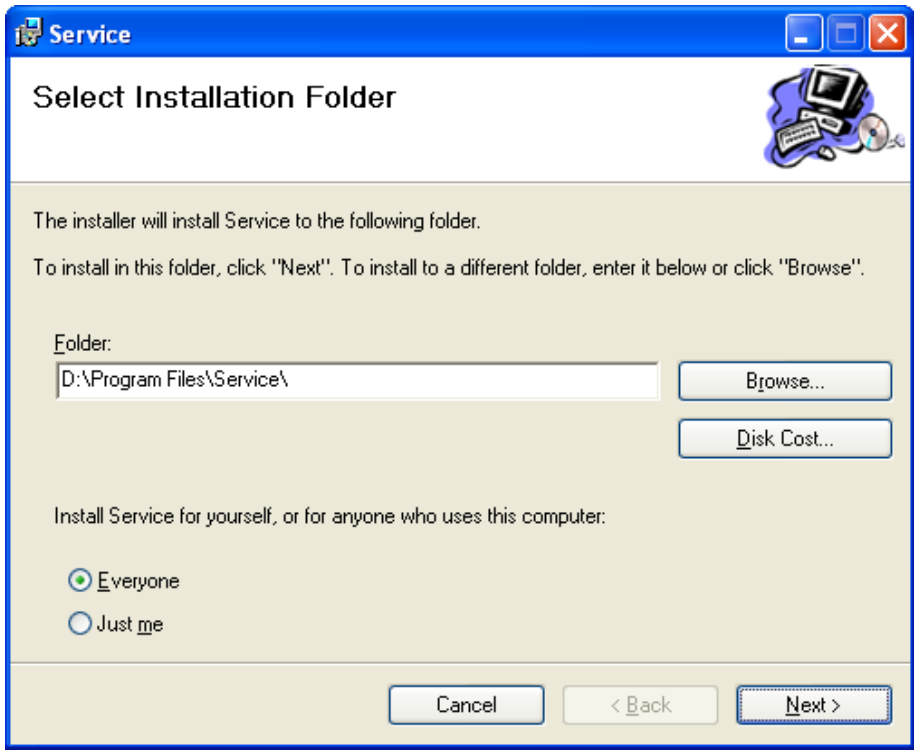
– příkaz na přidání vybraného přístroje do seznamu přístrojů

Příloha B – Instalace aplikace Service

K vytvoření instalátoru aplikace Service jsme použil šablonu *Setup Project* z modulu *Setup and Deployment* v programovacím prostředí *Microsoft Visual Studio.* Tato šablona zahrne do instalačního souboru všechny knihovny potřebné pro spuštění nainstalované aplikace.

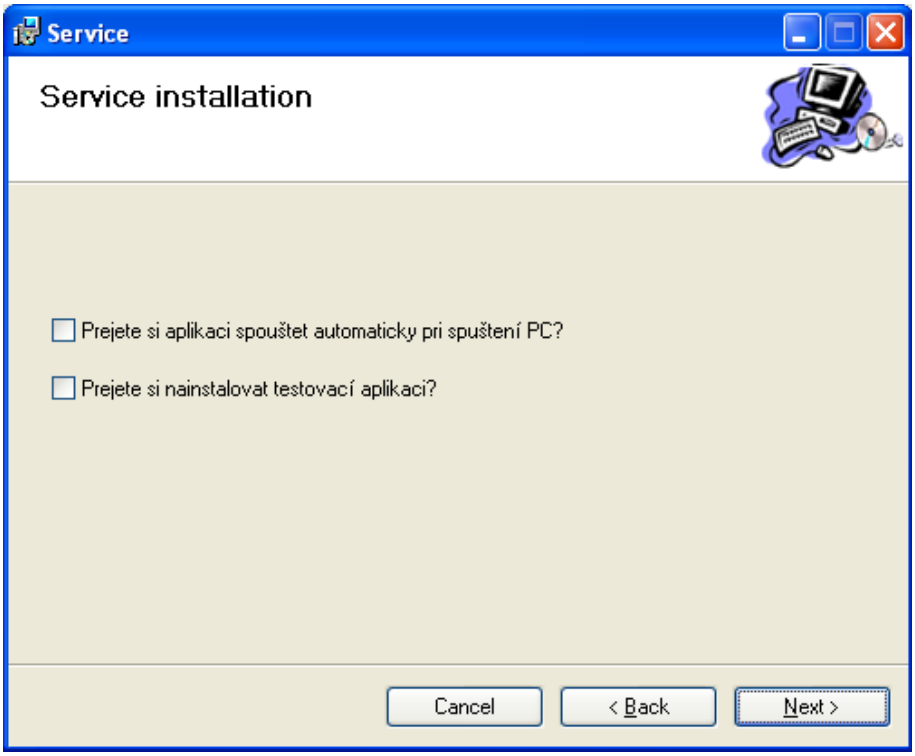
Instalace se spustí vytvořeným instalačním souborem. Samotný postup instalace je znázorněn na následujících obrázcích:

Na Obr. A.1 uživatel vybírá místo, na které má být aplikace v počítači nainstalována.



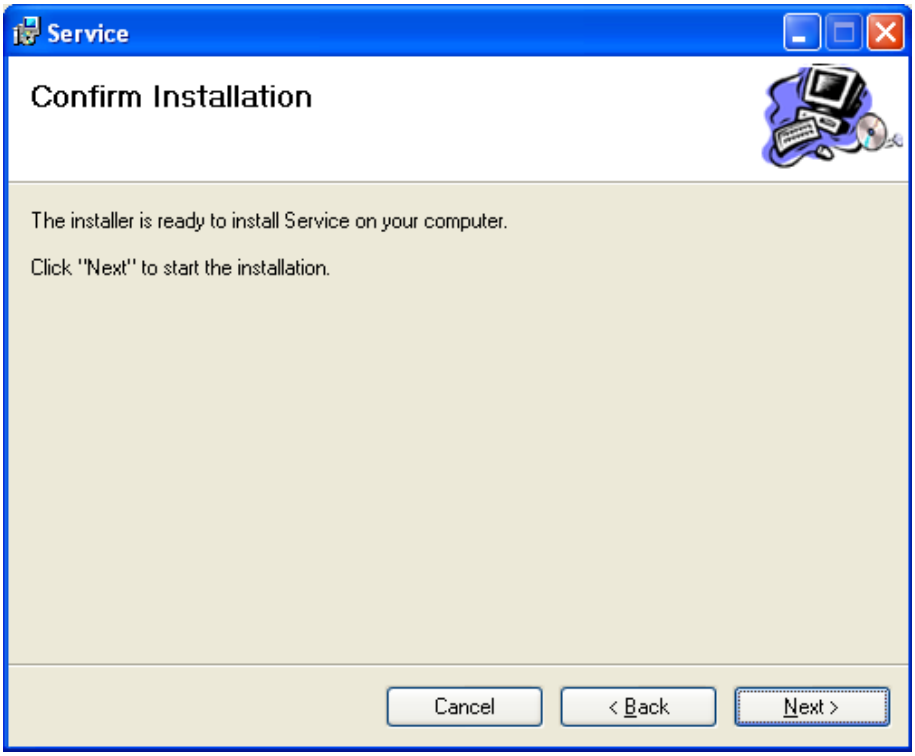
Obr. A.1: Výběr umístění

Následující okno instalace (viz Obr. A.2) dává uživateli na výběr, zda chce aplikaci Service spouštět automaticky při každém spuštění počítače. Druhou volbou v okně je možnost nainstalovat spolu se Service i testovací aplikaci, která slouží pro komunikaci se Service.

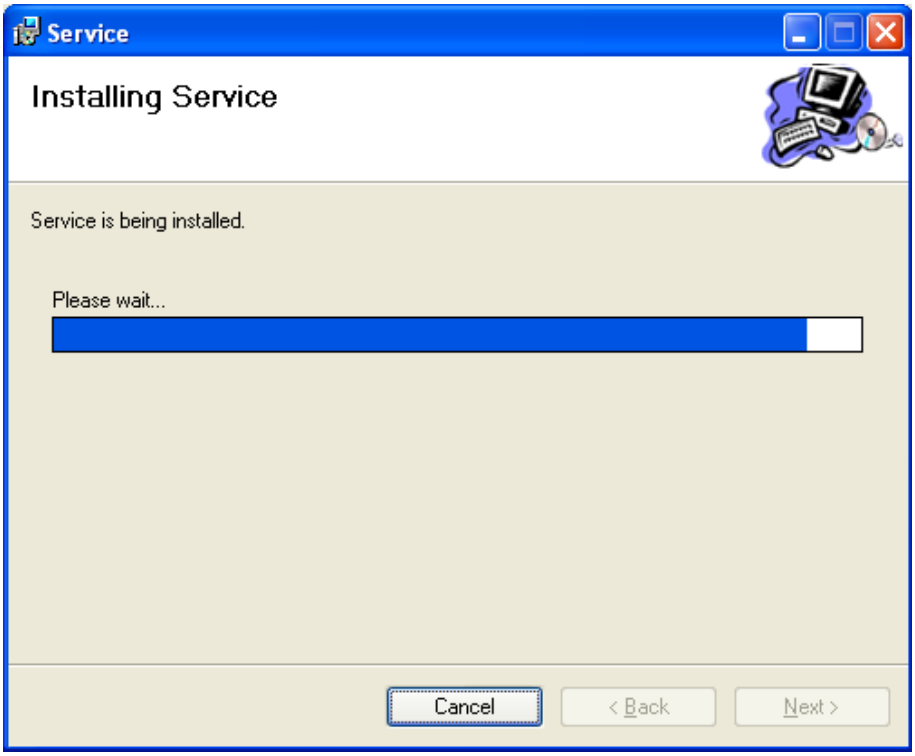


Obr. A.2: Doplňující volby

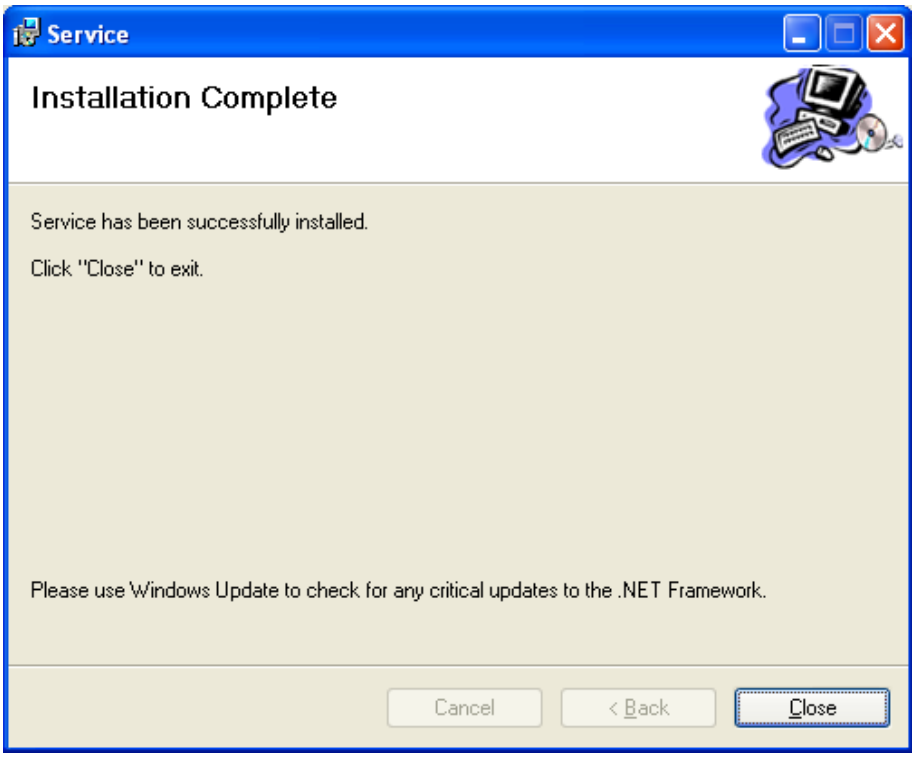
Následující tři obrázky znázorňují potvrzení instalace (viz Obr. A.3), dále samotný průběh instalace(viz Obr. A.4) a nakonec i její dokončení (viz Obr. A.5).



Obr. A.3: Potvrzení instalace



Obr. A.4: Průběh instalace



Obr. A.5: Dokončení instalace