VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2016 Bc. Matej Kišac

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

DISTRIBUOVANÉ APLIKACE S VYUŽITÍM FRAMEWORKU WINDOWS COMMUNICATION FOUNDATION

DISTRIBUTED APPLICATIONS USING WINDOWS COMMUNICATION FOUNDATION FRAMEWORK

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Matej Kišac

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Ing. Ivo Lattenberg, Ph.D.

BRNO 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**
Ústav telekomunikací

Student: Bc. Matej Kišac ***ID:*** 146857
Ročník: 2 ***Akademický rok:*** 2015/16

NÁZEV TÉMATU:

Distribuované aplikace s využitím frameworku Windows Communication Foundation

POKYNY PRO VYPRACOVÁNÍ:

Prostudujte základní principy distribuovaných aplikací. Prostudujte také možnosti frameworku Windows Communication Foundation. S využitím knihovny WCF realizujte distribuovanou aplikaci, pro hledání prvočísel. Pro možnosti srovnání vytvořte také jednoduchou aplikaci pro hledání prvočísel pracující na jednom počítači. Proveďte výkonostní srovnání Vaší aplikace na více počítačích s výkonem aplikace pracující jen na jednom počítači.

DOPORUČENÁ LITERATURA:

[1] WATSON, B., C# 4.0 - řešení praktických programátorských úloh, Zoner Press, 2010, 656 s., ISBN 978-8--7413-094-6

[2] VIRIUS, M., C# 2010 Hotová řešení, Computer Press, 2012, 424 s., ISBN 978-80-251-3730-7

[3] LOWY, J., Programming WCF Services, O'Reilly, 2007, ISBN 978-0596526993

Termín zadání: 1.2.2016 ***Termín odevzdání:*** 25.5.2016

Vedoucí práce: doc. Ing. Ivo Lattenberg, Ph.D.
Konzultant diplomové práce:

doc. Ing. Jiří Mišurec, CSc., *předseda oborové rady*

ABSTRAKT

Táto práca sa zaoberá distribuovanými aplikáciami a technológiou WCF. V prvej časti práce sa prostredníctvom teórie oboznamujeme s vlastnosťami distribuovaných systémov a ich základnými modelmi. Ďalšia časť je venovaná rozboru technológie WCF a popisu hlavných stavebných prvkov každej WCF aplikácie. Nasledujúca kapitola má za úlohu priblížiť problematiku faktorizácie čísel. Implementácia poznatkov o technológii WCF je následne spracovaná v štvrtej časti, ktorá tiež ukazuje možnosti WCF pri návrhu servisne orientovaných aplikácií. Záver obsahuje návrh riešenia distribuovanej aplikácie na faktorizáciu čísel a porovnanie výkonu s aplikáciou dedikovanou pre jedno zariadenie.

KĽÚČOVÉ SLOVÁ

distribuovaný systém, Windows Communication Foundation, WCF služba, WCF klient, WCF hosťovanie, faktorizácia čísla

ABSTRACT

This thesis deals with distributed applications and WCF framework. The first part is based on theoretical information about distributed systems and we also concentrate on models of distributed systems. Next part describes WCF framework and key elements of WCF application. The following chapter is designated to introduce information about prime factorization. Then the knowledge from previous parts is used to create examples of service-oriented applications. In conclusion we discuss main parts of designing distributed application to solve factorization problem. Finally the comparison of distributed and dedicated application is made.

KEYWORDS

distributed system, Windows Communication Foundation, WCF service, WCF client, WCF hosting, prime factorization

KIŠAC, Matej *Distribuované aplikace s využitím frameworku Windows Communication Foundation*: diplomová práca. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2015/2016. 62 s. Vedúci práce bol doc. Ing. Ivo Lattenberg, Ph.D.

PREHLÁSENIE

Prehlasujem, že som svoju diplomovoú prácu na tému „Distribuované aplikace s využitím frameworku Windows Communication Foundation“ vypracoval samostatne pod vedením vedúceho diplomovej práce, využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej diplomovej práce ďalej prehlasujem, že v súvislosti s vytvorením tejto diplomovej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), vo znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákoníka č. 40/2009 Sb.

Brno ....

(podpis autora)

POĎAKOVANIE

Rád by som poďakoval pánovi doc. Ing. Ivovi Lattenbergovi Ph.D. za jeho ochotu a pomoc pri riešení tejto práce.

Brno . . . . . .

. . . . . .

(podpis autora)

SIX research centre
sensor, information and communication systems

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic

http://www.six.feec.vutbr.cz

POĎAKOVANIE

Výskum popísaný v tejto diplomovej práci bol realizovaný v laboratóriách podporených z projektu SIX; registračné číslo CZ.1.05/2.1.00/03.0072, operačný program Výskum a vývoj pre inovácie.

Brno ....

(podpis autora)

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

2007–13
OP Výzkum a vývoj pro inovace

OBSAH

ZOZNAM OBRÁZKOV

ÚVOD

Distribuované aplikácie nám v súčasnosti prinášajú obrovské možnosti v riešení nadmerne zložitých úloh a pomáhajú tým zástupu laboratórnych výskumov v rôznych odvetviach. Preto má rozvoj technológií v oblasti distribuovaných výpočtov veľký potenciál. V minulosti bolo riešenie aplikácií založené na voľbe konkrétnej technológie, kde kooperácia s ostatnými predstavovala výzvu v jej návrhu. V súčasnosti môžeme s výhodou využívať technológie ako WCF, ktoré nás oslobodzujú od riešenia problémov s interoperabilitou. Práca sa preto zameriava na vysvetlenie základných princípov distribuovaných systémov a WCF frameworku.

Úlohou tejto práce je predstavenie možností technológie WCF, ktorá je s výhodou využívaná pri návrhu distribuovaných aplikácií. V prvej časti sa čitateľ dozvie bližšie informácie o distribuovaných systémoch ako takých, a s tým spojených kladoch a komplikáciách, ktoré sa s ich návrhom spájajú. Časť práce bude venovaná technológii WCF, ktorá je ideálnou voľbou pre tvorbu aplikácií založených na komunikácii väčšieho množstva výpočtových prvkov. Táto kapitola predstavuje akýsi prienik do tvorby WCF aplikácií, kde sa môžeme stretnúť s podrobným výkladom zvoleného postupu pri riešení úloh, ktorých hlavnou myšlienkou je dôraz na jednoduchosť a najmä efektivitu realizácie úlohy.

Po získaní skúseností s návrhom jednoduchých aplikácií pristúpime k realizácii distribuovanej aplikácie. Podstata tejto aplikácie bude vychádzať z teoretických poznatkov uvedených v kapitole, ktorá odhalí základné možnosti riešenia faktorizácie čísel. Na základe vhodne zvolenej metódy faktorizácie bude vytvorená aplikácia, prostredníctvom ktorej bude užívateľ schopný rozdistribuovať túto úlohu medzi zariadenia, ktoré sa prihlásia ako odberatelia našej služby. Časť určená distribuovanej aplikácii objasní jednotlivé fázy návrhu. V prvej fáze bude bližšie odôvodnená voľba vhodnej metódy faktorizácie na distribúciu problému. Druhá fáza bude zameraná na predstavenie základnej architektúry distribuovanej aplikácie, kde nájdeme bližší popis základných prvkov, ktoré sa budú podieľať na riešení. Tretia fáza sa bude zaoberať spôsobom, akým komunikujú medzi sebou prvky aplikácie a vecnými radami, ktoré súvisia s uvedením programu do behu. Ďalej budú prezentované funkcie navrhnuté na riešenie problému v zjednodušenej forme, ako aj v podobe detailnejších poznámok. Na záver použijeme vytvorené aplikácie k overeniu kladného výkonnostného prínosu distribuovanej aplikácie v porovnaní s aplikáciou používanou jedným zariadením.

1 DISTRIBUOVANÝ SYSTÉM

1.1 Úvod

Pri aktuálnom vývoji technológií dochádza k neustálemu zvyšovaniu nárokov na hardware zariadení, ktoré na riešenie vedeckých problémov vyžadujeme. V prípade skutočne komplexných úloh nie sme schopní dosiahnuť potrebný výkon v rámci jedného zariadenia, ktorý by umožňoval splnenie danej úlohy v akceptovateľnom čase. Takéto problémy podmienili vznik distribuovaných systémov. Distribuovaný systém tak využíva vhodné rozdelenie tejto zložitej úlohy medzi procesory viacerých počítačov, ktoré navzájom komunikujú.

Distribuovaný systém tak môžeme definovať ako:

systém, v ktorom sa množina nezávislých počítačov javí ako jeden samostatný, kde vzájomná súčinnosť je riadená obojstrannou výmenou správ. Samotná komunikácia zariadení je podmienená prepojením komunikačným médiom a softwarom, ktoré sú schopné koordinovať operácie a zabezpečiť zlúčenie použitých prostriedkov (hardware, data) [1].

1.1.1 Dôležité aspekty distribuovaného systému

- **Nesúrodosť:** podmienená použitím rôznych programovacích jazykov, rôzne typy sietí a hardwaru vyžaduje použite tzv. middleware vrstvy, ktorá zabezpečí celkovú abstrakciu [1]
- **Otvorenosť:** ľubovoľný systém musí byť schopný komunikovať s ktorýmkoľvek iným systémom, pričom rôzne zdroje využívajú unifikované rozhranie pre komunikáciu [1]
- **Škálovateľnosť:** možnosť pridania nových zdrojov do jestvujúceho systému pri zachovaní efektivity bez nutnosti zásahov do algoritmov systému [1]
- **Zabezpečenie:** zdrojov informácií (šifrovanie, autentizácia, autorizácia, atď.) [1]

1.1.2 Klady distribuovaných systémov

- Spoločné používanie dát alebo zariadení (databázy letov v aerolíniách, tlačiarne) [1]
- Výkon (pri vzájomnej spolupráci dostatočného počtu zariadení môžeme aj pri nízkom výkone jednotlivcov dosiahnuť výpočtovej sily vysoko špecializovaného a drahého zariadenia) [1]

- Flexibilita (distribuovaný systém je schopný chodu aj za zlyhania jednotlivých klientskych zariadení) [1]
- Jednoduchá správa systému (aktualizácia systému, zálohovanie dát) [1]

1.1.3 Zápory distribuovaných systémov

- Ovplyvniteľné parametrami a spoľahlivosťou siete (šírka pásma, prenosová rýchlosť a oneskorenie) [1]
- Bezpečnosť (viac klientskych staníc pracujúcich na danom výpočte predstavuje bezpečnostnú hrozbu, a preto je potrebné dbať pri návrhu systému na tento podstatný faktor) [1]
- Zložitosť návrhu a samotná implementácia (úlohy distribuovaného systému musia byť optimálne realizované, aby nedochádzalo k chybám, ktoré sú v komplexnom systéme ťažko lokalizovateľné a stabilita systému je taktiež dôležitá pre správnu funkcionalitu bez výpadkov klientskych zariadení) [1]

1.2 Modely Distribuovaných systémov

Na základe rôznych aspektov môžeme definovať tzv. modely distribuovaných systémov. Tieto modely nám dávajú priestor k pochopeniu rôznych implementácií systémov, možnosť nájsť slabé a silné stránky sledovaného systému, a aj možnosť klasifikácie systémov. Hovoríme tak o modeloch [2]:

- Popisujúcich architektúru systému
- Vzájomnej interakcie
- Riešenia chybových stavov

1.2.1 Modely popisujúce architektúru systému

Hlavnou úlohou modelu je popis rozmiestnenia prvkov systému a operácií, za ktoré sú prvky zodpovedné. Predstaviteľmi týchto modelov sú:

Client-Server model

Procesy distribuovanej úlohy sú vykonávane servermi, ktoré poskytujú služby pripojeným klientom. Typicky je model založený na jednoduchej komunikácii otázka/odpoveď, prípadne na vzdialenom volaní procedúr (RPC). Špeciálnym prípadom je komunikácia servera so serverom, ktorá je tiež možná za podmienky, že jeden z účastníkov (žiadateľ) vystupuje v danej situácii ako klient [3].

Peer-to-peer model

Model stelesňujúci myšlienku rovnocennosti prvkov systému. V tomto prípade nerozlišujeme klientskú a serverovú úlohu v systéme. Komunikácia prebieha priamo medzi prvkami a jej priebeh je podmienený implementáciou prípadu. Charakteristickým prvkom je tiež spoločné používanie dát, kde každý prvok disponuje určitou časťou vo vlastnej pamäti[3].

Kombinované modely

Modely, ktoré majú pôvod v predchádzajúcich dvoch základných myšlienkach. Jednou z možných variácií týchto systémov je **proxy server** (server, ktorý má za úlohu spracovanie klientskych žiadostí o kópiu zdrojov, ktoré sú umiestnené na iných serveroch). Typické využitie proxy serveru spočíva v použití ako cache pamäte pre webové servery, kde dochádza k dočasnému uloženiu odpovedí na požiadavky o webové stránky, obrázky, atď. Tak dokážeme zjemniť výkonnostné požiadavky a zvýšiť dostupnosť serveru, ktorý danú službu zastrešuje. Ďalším významným faktorom je zníženie nárokov na prenosové cesty, ktoré ku komunikácii využívame. Druhou možnosťou je použitie tzv. **mobilných agentov**: títo k svojej činnosti využívajú mobilný kód, ktorý je schopný vykonávať cielenú funkciu na aktuálne používanom zariadení. Kód je schopný transportu z jedného prvku do druhého v rámci siete, a tak získava prístup k veľkej škále zdrojov rôznych zariadení. Model je s obľubou využívaný na zber dát a hromadné inštalácie softwaru [4].

1.2.2 Modely vzájomnej interakcie

Časová synchronizácia (procesov, doručovania správ) zariadení predstavuje problém v rámci implementácie do distribuovaných systémov. Riešením týchto ťažkostí sú dva základné modely [2]:

Synchrónne distribuované systémy

Sú systémy, u ktorých presne definujeme limitné body:

- Vykonania špecifického procesu
- Odoslania a príjmu správ
- Rozdielu posunu lokálnych hodín od reálneho času

Implementácia synchrónnych systémov je veľmi náročná, s čím sa spájajú aj zvýšené náklady na tvorbu týchto systémov [2].

Asynchrónne distribuované systémy

Predstavujú opačný extrém v porovnaní so synchrónnymi systémami. Jedná sa o prípady, kedy nie sú použité žiadne hraničné časy vykonania konkrétneho procesu, odosielania a príjmu správ, z toho vyplýva nemožnosť časovej predikcie chovania systému. Za typického predstaviteľa môžeme považovať Internet, kde napr. u FTP prenosu dát nedokážeme presne určiť celkový čas prenosu súborov. Asynchrónne systémy sú tak v súčasnosti širokospektrálne využiteľné a implementované do množstva riešení [2].

1.2.3 Modely riešenia chybových stavov

Na chovanie distribuovaných systémov má vplyv software aj hardware. Obe tieto časti môžu byť ovplyvnené množstvom chybových stavov, ktoré sú následne pôvodcom nesprávnych výsledkov, alebo zlyhania funkcie systému. Ďalším významným faktorom je prenosový kanál, kde chyby prenosu spôsobujú komplikácie chovania systému. Preto považujeme za nutné zaoberať sa modelmi na riešenie možných zlyhaní systému. Medzi spomínané modely patria [2]:

Chyba vynechaním

Je charakteristická chybou vo výkone procesu v procesore alebo chybou prenosového kanálu, čo má za následok nevykonanie danej operácie.

Byzantské chyby

Model, u ktorého môže dôjsť k vzniku ľubovoľnej chyby ako zo strany procesoru tak prenosového kanálu. Jedná sa tak o najhorší možný prípad chovania distribuovaného systému. Typickým prejavom systému je vynechanie očakávaného procesu a komunikácie alebo naopak vykonanie nevyžiadaného.

Chyba časovača

Je definovaná prekročením určených časových limitov vykonania procesu, komunikácie alebo časového posunu. Model je tak využiteľný len pre synchrónne distribuované systémy, u ktorých sú hodnoty vyššie zmienených limitov špecifikované.

2 WINDOWS COMMUNICATION FOUNDATION (WCF)

Koncept distribuovaného programovania je už dlhé obdobie veľmi aktuálnou témou vývoja aplikácií. K realizácii návrhu tohto konceptu boli vytvorené a následne využívané rôzne technológie ako ASMX (ASP.NET Web Service), WSE (Web Service Enhancements), NET Remoting, atď. Rôzne technológie však priniesli nemožnosť interoperability. To malo za následok snahu Microsoftu o preklenutie týchto problémov sofistikovanejším riešením, ktorým sa stala platforma WCF postavená na princípe service-oriented architecture (SOA). S príchodom WCF došlo k zjednoteniu predchádzajúcich technológií, a tým aj k zjednodušeniu návrhu distribuovaných aplikácií.

2.1 Výhody použitia WCF

Ako už bolo spomenuté, WCF prináša do prostredia vývoja aplikácií orientovaných na služby množstvo výhod a tými najvýraznejšími sú [5]:

- **Široké spektrum možností programovacieho modelu:** variabilita riešení návrhov aplikácií, obmedzenie rozsahu kódu na minimum.
- **Interoperabilita:** WCF prináša zjednotené prostredie, kde je užívateľ schopný využívať výhody všetkých starších technológií, a taktiež prináša nezávislosť na použitej platforme (WCF služby tak môžu byť využité napr. v PHP, Jave, atď.).
- **Možnosti riešenia komunikácie služieb:** WCF na základe nezávislosti komunikácie od samotnej služby zabezpečuje slobodu v použití rôznych komunikačných protokolov, vývojár tak podľa potrieb môže jednoducho implementovať protokoly ako HTTP, TCP, UDP, NamedPipe a mnoho ďalších.
- **Bezpečnosť a spoľahlivosť:** v porovnaní so staršími technológiami ako ASMX, kde bolo nutné nájsť rovnováhu medzi trojicou (bezpečnosť, spoľahlivosť a výkon), prináša WCF riešenie, v ktorom sa problematikou znižovania výkonu na úkor vyššie uvedených vlastností viac nemusíme trápiť. Za ďalšiu z výhod WCF môže byť považované odľahčenie spôsobu zavedenia bezpečnostného modelu do aplikácií.

2.2 Service-oriented architecture (SOA)

SOA môžeme považovať za návrhový vzor používaný k vývoju takých služieb, ktoré sú navzájom autonómne, ale majú schopnosť vzájomnej komunikácie prostredníc-

tvom správ. Na základe samostatnosti komponentu SOA architektúry je systém jednoducho rozšíriteľný, nahraditeľný a spravovaný. V architektúre orientovanej na služby tak musíme dbať na dodržanie základných princípov, ktoré za týmto návrhovým vzorom stoja [6]:

- **Služby majú explicitné hranice:** klient je zbavený potreby poznania implementácie ostatných služieb, ku ktorým pristupuje a komunikuje. Nezáleží tak na použitej technológii a vnútorných procesoch využívanej služby. V prípade zmeny služby teda nie je nutné meniť chovanie klienta.
- **Služby sú autonómne:** najmä v prostredí distribuovaných aplikácií je prirodzeným faktorom vývoj. Preto je nutné, aby služby ako také fungovali samostatne, bez vplyvu ostatných služieb. Úprava správania služby nemení spôsob fungovania celého systému.
- **Kompatibilita služieb je založená na politike:** v SOA je veľký dôraz kladený na komunikáciu služieb. Počiatok vzájomnej komunikácie je postavený na výmene informácií o úrovni kompatibility služieb. Tento údaj je obsiahnutý v tzv. politike chovania danej služby, ktorá popisuje jednoznačné schopnosti služby a spôsob komunikácie. V prípade zhody záujmov politík vzniku komunikácie medzi službami nič nebráni. V opačnom prípade je však žiadosť zamietnutá. Výhodou politík je tak schopnosť umožniť migráciu služby z jedného prostredia do druhého bez nutných úprav chovania samotnej služby.
- **Spoločné využívanie schém a kontraktov:** služby využívajú ku komunikácii dátové schémy a kontrakty. K definícii schém a kontraktov využívame XML, ktoré zaisťuje nezávislosť na použitej platforme. Schémy sú využité pre pochopenie dátovej štruktúry, pričom kontrakty definujú chovanie služby.

Komunikácia aplikácií vytvorených s použitím WCF prebieha výhradne prostredníctvom SOAP (Simple Object Access Protocol) protokolu. SOAP protokol sprostredkováva užívateľovi správy so štruktúrovanými informáciami založenými na XML, ktoré fungujú ako komunikácia medzi službou a klientom [6].

2.3 Hlavné komponenty WCF

Každá WCF aplikácia pozostáva z troch základných stavebných komponentov, a tými sú:

- Služba
- Hosťovanie služby
- Klient

2.4 WCF služba

Vytvorenie služby je kľúčovým bodom vzniku WCF aplikácie. Aplikácia pozostáva z jednej alebo obvykle z viacerých služieb, ktoré na základe SOA môžu byť realizované pri použití rôznych technológií. Služby sú entitami, ktoré zaobstarávajú špecifickú funkcionalitu klientom. Títo klienti ju následne vyžadujú ku svojej správnej činnosti. Každá WCF služba je zložená z jedného alebo viacerých endpointov, ktoré musia mať definovaný tzv. binding. Ďalšou nemenej dôležitou súčasťou endpointu je pridelený contract služby, ktorý súvisí s implementáciou a funkcionalitou konkrétnej služby. Štruktúru WCF aplikácie kompletizuje popis služby, ktorého hlavnou úlohou je šírenie informácií o možných funkcionalitách služby a spôsobe, akým je možné k službe pristupovať. Šírenie popisu služby je typicky zabezpečené použitím štandardov, ako je napr. WSDL (Web Service Description Language) [7].

2.5 Endpoint

Endpointy sú nevyhnutnými prvkami služby, ktoré môžeme charakterizovať ako prostredníkov medzi klientmi a službami. Úloha endpointu tiež spočíva v určovaní typu správ a spôsobu komunikácie. Aby bol endpoint kompletný, musí zahŕňať tri hlavné súčasti, ktorými sú [7]:

- Address
- Binding
- Contract

2.5.1 Address

Unikátna adresa je neoddeliteľnou súčasťou každého vytvoreného endpointu, bez adresy by totiž nebolo možné lokalizovať endpoint klientskou aplikáciou. Vo WCF rozlišujeme primárne tri základné druhy adries [5]:

- **Adresa endpointu:** ako už bolo diskutované vyššie, jedná sa o jedinečnú adresu pridelenú konkrétnemu endpointu. Táto adresa je následne využitá klientom, ktorý s jej pomocou môže pristupovať k endpointu. Do konca relácie tak komunikuje so službou cez priradený endpoint. Adresa môže mať tvar napr. http://localhost:8080/Service/.
- **Základná adresa:** je voliteľnou možnosťou a umožňuje priradiť službe primárnu adresu, ktorú následne distribuuje na všetky vytvorené endpointy, čím dáva možnosť využívať len relatívne názvy endpointu.

- **MEX adresa:** je špeciálna adresa, prostredníctvom ktorej je klient schopný zistiť podstatné informácie o službe, tieto informácie sú prenášané v podobe služobných metadát. Príklad MEX adresy je http://localhost:8080/Service/mex.

Prvou významnou informáciou adresy je použitá transportná schéma. Nasledujúcim údajom každej adresy je názov počítača alebo domény, kde služba vykonáva svoju činnosť. Voliteľnou položkou adresy je číslo portu, ktorým sa priraďuje dátový tok k službe. Každá zo schém alebo protokolov, ktoré je možné využívať, má v predvolenom nastavení pridelené porty, používané v prípade, že bližšie nešpecifikujeme číslo portu pri návrhu služby. Poslednou súčasťou adresy je cesta definujúca endpoint. Na základe popisu vyššie môžeme hovoriť o obecnom formáte adries:

$$\text{protokol/transportnéschema} : //[\text{názovpočítača}|\text{domény}][: \text{port}]/\text{cesta} \qquad (2.1)$$

Výhodou WCF je skutočnosť, že podporuje veľké množstvo rôznych transportných protokolov a schém. Z hľadiska návrhu WCF služieb potom využívame nasledujúce možnosti [5]:

- **HTTP/HTTPS** (http://[názovpočítača /doména][:port (voliteľný) alebo HTTP:80, HTTPS:443])
- **TCP** (net.tcp://[názovpočítača/doména][:port (voliteľný)alebo predvolený 808])
- **IPC** (net.pipe://[názovpočítača/doména]/služba):akceptuje len volania z definovaného názvu počítača
- **MSMQ** (net.msqm://[názovhostiteľa]/[private]/typfronty)
- **Peer2Peer** (net.p2p://[názovhostiteľa]/služba)

2.5.2 Binding

Je povinnou súčasťou každého endpointu. Hlavnou úlohou bindingu je ustanoviť „Ako“ sa bude endpoint chovať pri komunikácii so žiadateľmi o službu, toto chovanie je charakterizované transportnou, bezpečnostnou a komunikačnou (kódovanie) schémou. WCF nám prináša možnosť využiť deväť preddefinovaných bindingov. Každý disponuje jedinečnými vlastnosťami, ktoré ho predurčujú na konkrétny druh služby. V prípade potreby je možné vstavané bindingy upravovať, alebo dokonca vytvoriť svoj vlastný. V tabulke 2.1 môžeme vidieť špecifické rysy preddefinovaných bindingov používaných vo WCF [7].

2.5.3 Contract

Je poslednou významnou časťou mozaiky pochopenia konceptov využívaných pri návrhu WCF služieb. Prostredníctvom contractov sú služby schopné uverejňovať podporované typy operácií a používaných štruktúr, ktoré budú v priebehu celej

Tab. 2.1: Tabuľka preddefinovaných bindingov.

Binding Type	Transport	Encoding	Interoperable	Duplex	Session
BasicHttpBinding	HTTP/HTTPS	Text, MTOM	Basic Profile 1.1	NO	NO
WSHttpBinding	HTTP/HTTPS	Text, MTOM	WS	NO	YES
WSDualHttpBinding	HTTP/HTTPS	Text, MTOM	WS	YES	YES
WSFederationHttpBinding	HTTP/HTTPS	Text, MTOM	WS	NO	YES
NetTcpBinding	TCP	Binary	.NET	YES	YES
NetPeerTcpBinding	P2P	Binary	Peer	YES	NO
NetNamedPipesBinding	NamedPipes	Binary	.NET	YES	NO
NetMsmqBinding	MSQM	Binary	.NET	NO	NO
MsmqIntegrationBinding	MSQM	-	MSQM	NO	NO

komunikácie využívané. Zabezpečujú tak interoperabilitu, ktorá je vyžadovaná pri komunikácii služby s klientom. Vo WCF sú prakticky používané štyri hlavné druhy contractov [7]:

- ServiceContract
- DataContract
- FaultContract
- MessageContract

ServiceContract

Je zodpovedný za popis operácií, ktoré sú zo strany služobného endpointu poskytované potenciálnym žiadateľom v podobe klientov. Klient tak získava prehľad o možnostiach využitia danej služby. Prehľad o špecifickej službe vychádza z nasledujúcich informácií [7]:

1. Dátové typy použité v správach
2. Umiestnenie operácie
3. Informácie o použitých protokoloch
4. Vzájomné previazanie operácií

Akým spôsobom môže vývojár ServiceContract aplikovať?

Vytvorenie ServiceContractu je jednoduché a to aplikáciou atribútu [ServiceContract] na konkrétnu triedu alebo rozhranie. Povedzme, že mám záujem vytvoriť službu ComplexService, ktorá nám umožní poskytovať základné matematické operácie s komplexnými číslami, ako je sčítanie, odčítanie atď. V prvom kroku je tak nutné vytvoriť predstaviteľa služby ServiceContract. Následne musíme našej službe priradiť jednotlivé operácie, ktoré bude poskytovať klientom. Operácie služby sú definované prostredníctvom atribútu [OperationContract], ktorý aplikujeme na metódy priradené vytvorenému rozhraniu. Vo výsledku bude mať ServiceContract nasledujúcu podobu [7]:

```
[ServiceContract]
public interface IComplexService
{
    [OperationContract]
    ComplexNumber add(ComplexNumber a, ComplexNumber b);
    [OperationContract]
    ComplexNumber sub(ComplexNumber a, ComplexNumber b);
}
```

Vytvorené rozhranie následne musíme implementovať na triedu, v ktorej budeme definovať chovanie všetkých implementovaných operácií.

DataContract

Môžeme považovať za popis dát, ktoré sú vymieňané medzi klientom a službou počas relácie v podobe metadát. Pred využívaním konkrétnej služby je nutné, aby sa klient a služba dohodli na forme dát, s ktorými sa ďalej bude pracovať. Implementácia je opäť založená na použití atribútu [DataContract] na triedu alebo dátový typ enum. Následne však musíme uviesť členov spadajúcich pod daný DataContract a toho docielime za pomoci atribútu [DataMember], ktorý umiestnime pred jednotlivé vlastnosti triedy. Dôležitým aspektom verzií .NET 3.5 a vyšších je, že atribúty [DataContract] a [DataMember] sú automaticky vytvorené u všetkých vlastností označených ako „public“ bez nutnosti špecifikácie tejto skutočnosti v kóde. Tento fakt platí aj pre primitívne dátové typy (int, string, atď.). Príklad vytvorenia datacontractu pre aplikáciu uvedenú v predchádzajúcej časti môžeme vidieť na tomto mieste [7]:

```
[DataContract]
public class ComplexNumber
{
    [DataMember]
    public int RealPart
    {
        get { return realPart; }
        set { this.realPart = value; }
    }
    [DataMember]
    public int ImaginaryPart
    {
        get { return imaginaryPart; }
        set { this.imaginaryPart = value; }
    }
}
```

MessageContract

Spĺňa výhradne úlohu správcu formátovania SOAP (Simple Object Access Protocol) správ. K použitiu MessageContractu pristupujeme typicky až pri službách, ktoré kladú dôraz na interoperabilitu pri využití neštandardných komunikačných schém.

Vo väčšine prípadov však stačí importovať DataContract, ktorý poskytne dostatočnú kontrolu nad komunikáciou klienta a služby. V rámci MessageContractu tak získavame väčšiu kontrolu nad formátom správ, a to najmä špecifikáciou informácií, ktoré bude obsahovať telo a hlavička SOAP správy. Definícia je opäť založená na použití atribútov [MessageContract], [MessageHeader] a [MessageBodyMember] [7].

FaultContract

V prípade aplikácií orientovaných na služby je treba dbať na veľké množstvo rôznych výnimiek alebo chýb, ku ktorým pri komunikácii alebo výkone služby môže dôjsť. Keď sa však takéto komplikácie na strane služby objavia, je vhodné, aby sa klient o nich dozvedel a mohol tak vhodným spôsobom reagovať. WCF nám umožňuje realizovať vlastné silné typové výnimky a za pomoci atribútu [FaultContract], aplikovaného na vytvorenú triedu, je služba schopná špecifikovať zhluk potenciálnych chýb, ktoré môžu nastať a následne sa šíriť na stranu klienta. Riešenie [FaultContract] je založené na reprezentácii technologicky špecifických výnimiek prostredníctvom štandardu SOAP, ktorý vnáša neutralitu do chybových hlásení [7].

2.6 Hosťovanie služby

V predchádzajúcej časti boli popísané dôležité aspekty tvorby služieb vo WCF, avšak aby sme službu dostali do chodu, je nutné vytvoriť Windows hostiteľský proces. Tento proces je následne schopný sprostredkovať hosťovanie viacerým službám, a tak sprostredkováva klientovi možnosť prístupu k požadovanej službe. Pre tvorbu hostiteľského procesu máme nasledujúce možnosti [7]:

- Internet Information Services (IIS)
- Hosting in a Console or Desktop application (Self-hosting)
- Windows Activation Service (WAS)
- Hosting in Windows Service

2.6.1 Internet Information Services

Hosťovanie služby v IIS prináša jednu zásadnú výhodu, a to v podobe automatického spustenia hostiteľského procesu pri prvej žiadosti klienta o danú službu. IIS je následne schopné monitorovať priebeh procesu a riešiť tak prípadné zlyhania automaticky prostredníctvom recyklovania procesu, čím jednoducho docielime obnovu systémových zdrojov. IIS tak môžeme považovať za akéhosi správcu životného cyklu

celej služby, ktorý k svojej funkcionalite využíva výhradne HTTP protokol. Nasadenie IIS je jednoduché s použitím .svc súboru, ktorý je zodpovedný za vytvorenie hosťovania [7].

```
<%@ ServiceHost
Language = "C#"
Debug = "true"
Service = "MyService"
%>
```

2.6.2 Hosting in a Console or Desktop application

Na rozdiel od IIS u Self-Hostingu je spôsob implementácie hosťovania definovaný samotným vývojárom. Vývojár musí v kóde definovať inicializáciu hosťovania služby, taktiež sa musí zaoberať kontrolou životného cyklu hostiteľského procesu. Self-hosting tiež prináša slobodu použitia rôznych komunikačných protokolov. Prostredie, v ktorom bude náš hostiteľ existovať, môže byť akýkoľvek Windows proces ako napr. Konzolová aplikácia, Windows Forms aplikácia, WPF aplikácia a iné. Spustenie reagovania služby na požiadavky klientov môžeme zabezpečiť pomocou triedy ServiceHost, konkrétne metódy Open. Ukončenie služby je potom riešené metódou Close [7].

2.6.3 Windows Activation Service

WAS hosťovanie prináša do návrhu prostredie s väčšiu flexibilitou a jednoduchým riadením služby. Je považované za akúsi nadstavbu IIS, a preto sú vlastnosti a výhody veľmi podobné. Dôležitú pridanú hodnotu však prináša možnosť použitia rôznych transportných protokolov (TCP, MSMQ, atď.) [7].

2.6.4 Hosting in Windows Service

Windows service je blízky akejkoľvek inej aplikácii bežiacej v prostredí Windows s niekoľkými charakteristickými rozdielmi. Windows service typicky beží na pozadí bez nutnosti implementácie užívateľského rozhrania. Výhodou Windows service je, že môže byť nastavená na automatické spustenie pri štarte systému, čo umožňuje klientom okamžitý prístup k službe. Taktiež poskytuje široké možnosti využitia rôznych transportných protokolov, a preto je medzi vývojármi možnosť hosťovania WCF služby pomocou Windows service možným variantom riešenia problému [7].

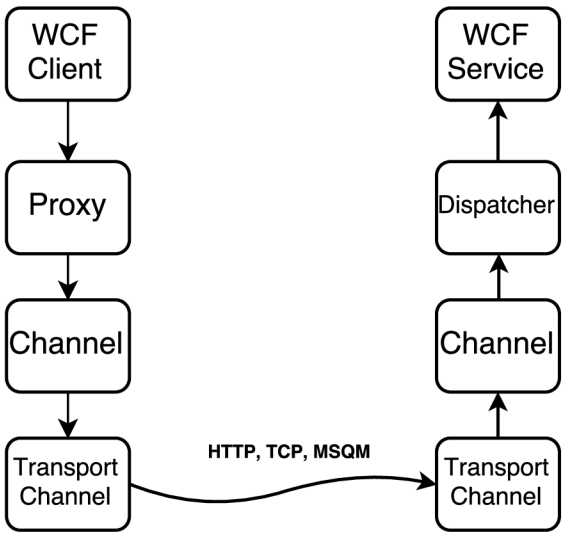
2.7 WCF Klient

Posledným neoddeliteľným komponentom WCF aplikácie je konzument funkcionalít služieb, ktorým je klientska aplikácia. WCF klient by nebol schopný využívať ope-

rácie bez zistenia informácií o službe, ku ktorej pristupuje, a preto je nutné uviesť adresu, na ktorej môžeme pristupovať k endpointu, aby mohlo dôjsť ku komunikácii klienta a služby. Spôsob komunikácie potom určíme zadaním bindingu na strane klienta a pre získanie povedomia o funkcionalitách je nutné uviesť aj contract služby. Klient následne môže byť riešený na ľubovoľnej platforme (WPF, Windows Forms, Console, WebForm, atď.) [5].

2.7.1 Štruktúra WCF klienta

Pri bližšom pohľade na WCF klienta berieme do úvahy, že k volaniu operácií a ku komunikácii so službou dochádza cez proxy triedu. Proxy trieda reprezentuje použitý ServiceContract služby, ale zároveň obsahuje metódy, ktoré sú využívané k správe životného cyklu proxy a k riadeniu prístupu k službe. Ďalšou súčasťou WCF klienta je implementácia rozhrania IClientChannel. Rozhranie tak podporuje operácie prebiehajúce nad prenosovým kanálom, ako ukončovanie klientskych relácií. Tieto operácie umožňujú vývojárom lepšiu kontrolu nad konkrétnym kanálom. Za poslednú časť považujeme komunikačný kanál, ktorý vznikne na podnet protokolov definovaných v bindingu. Obrázok 2.1 zobrazuje zjednodušený nákres štruktúry WCF klienta a služby, s ktorou komunikuje [5].



Obr. 2.1: Štruktúra komunikácie WCF klienta.

3 FAKTORIZÁCIA ČÍSEL

Predstavuje v oblasti aritmetiky jednoznačný proces rozkladu zloženého čísla na súčin tzv. faktorov (prvočísel). Problém faktorizácie veľkých čísel je značne využívaný pri realizácii kryptografických protokolov. Súčasné algoritmy spojené s faktorizáciou čísel neposkytujú dostatočnú efektivitu riešenia tohto problému v prípade zvolenia veľmi veľkých vstupných čísel. Pre prípad aktuálne známych algoritmov riešenia tohto problému hovoríme o subexponenciálnej až exponenciálnej zložitosti. Táto skutočnosť je úspešne využívaná v prípade asymetrického šifrovania RSA kryptosystémom, kde za bezpečné považujeme voľbu čísla s dĺžkou väčšou ako 2048-bitov. Medzi najčastejšie spôsoby riešenia faktorizácie čísel patria:

- Metóda postupného delenia
- Pollardova $p - 1$ metóda
- Pollardova ρ metóda
- Fermatova metóda

3.1 Metóda postupného delenia

Je implementačne najjednoduchším algoritmom využívaným k riešeniu faktorizácie čísel. Algoritmus tiež umožňuje overenie, že zadaná hodnota je zloženým číslom alebo prvočíslom. Princíp algoritmu vychádza zo znalosti čísla N určeného k faktorizácii, kde následná realizácia procesu prebieha postupným delením vstupnej hodnoty N všetkými prvočíslami P, pre ktoré platí $2 \leq P \leq \sqrt{N}$. Ak nájdeme v tomto rozsahu prvočíselného deliteľa, u ktorého platí nasledujúci výraz: $N \equiv 0 \,(\mathrm{mod}\, P)$, môžeme hodnotu N označiť ako zložené číslo a pokračovať v rozklade novej hodnoty $N = N/P$. V opačnom prípade môžeme hodnotu N označiť za prvočíslo. V nasledujúcich iteráciách postupujeme zhodne, pokiaľ nachádzame ďalšie prvočísla s nulovým zvyškom po delení. Nasledujúci príklad predstavuje možnosť implementácie algoritmu v prostredí C# [8].

```
public BigInteger trialDivision(BigInteger N)
{
    for (BigInteger i = 2; i <= SqRtN(N); i++)
    {
        if (N % i == 0)
        {
            return i;
        }
    }
    return N;
}
```

3.2 Fermatova metóda

Vychádza z reprezentácie zloženého čísla N ako rozdielu mocnín definovaného nasledovne:

$$N = a^2 - b^2 \tag{3.1}$$

z toho môžeme odvodiť, že

$$N = (a + b)(a - b) \tag{3.2}$$

Spôsob chovania Fermatovho algoritmu môžeme zhrnúť do niekoľkých jednoduchých krokov:

1. Definujeme hodnotu N určenú k faktorizácii
2. Získame druhú odmocninu čísla N, zaokrúhlime nahor k najbližšiemu celému číslu a označíme ako A
3. S využitím vzťahu 3.1 môžeme tvrdiť, že $B^2 = A^2 - N \Rightarrow B = \sqrt{A^2 - N}$
4. Následne v jednotlivých iteráciách inkrementujeme A v rozsahu $\left\langle \sqrt{N}, N \right\rangle$ až do bodu, kedy druhou odmocninou výsledku kroku 3 je celé číslo
5. So znalosťou A a B (kroky 3 a 4) môžeme tvrdiť, že prvočíselné faktory p_1 a p_2 vstupnej hodnoty N vyplývajú z rovnice 3.2 kde $p_1 = A - B$ a $p_2 = A + B$

V nasledujúcej ukážke vidíme možnosť aplikovania tejto metódy v prostredí C# [8].

```
public BigInteger fermatMethod(BigInteger N)
{
    BigInteger A,B;
    A = SqRtN(N)+1;
    B = A * A - N;
    while(!isSqrtBig(B))
    {
        A = A + 1;
        B = A * A - N;
    }

    return A - SqRtN(B);
}
```

3.3 Pollardova p−1 metóda

Môže byť zaradená medzi špeciálne faktorizačné metódy, a to najmä kvôli možnosti aplikácie len na určité typy čísel. Pollardovu metódu s výhodou môžeme používať v prípade čísel, ktoré majú unikátne delitele predstavujúce mocniny prvočísel menších než hranica B. Definujeme teda B-veľmi hladké prirodzené číslo, pre ktoré platí, že $p^i \leq B$, kde p je ľubovoľné prvočíslo a i je prirodzené číslo. Napríklad o čísle $2520 = 2^3 \cdot 3^2 \cdot 5 \cdot 7$ hovoríme, že je 7-hladké a zároveň 9-veľmi hladké ($2^3 = 8$, $3^2 = 9$). Princíp metódy je založený na Malej Fermatovej vete, ktorá vychádza z tvrdenia, že pre každé prvočíslo P nedeliteľné číslom A, platí: $A^{P-1} \equiv 1 \pmod{P}$. Z toho

môžeme usúdiť, že $A^P - A$ je deliteľné prvočíslom P. Pollardov $p-1$ algoritmus môžeme zhrnúť v nasledujúcich krokoch [9]:

1. Definujeme vstupné číslo N a hranicu B, ktorej hodnota môže byť nahradzovaná $\sqrt{N}$
2. Počítame $A^{k!}$ pre všetky $k = \langle 2, B \rangle$, kde v každej iterácii využívame krok 3
3. Nájdeme najväčší spoločný deliteľ hodnoty výrazu $(A^{k!} - 1) \bmod N$ a čísla N
4. V prípade nájdenia netriviálneho najväčšieho spoločného deliteľa získame faktor hodnoty N

Implementácia Pollardovej metódy v prostredí C# môže mať nasledujúcu formu:

```
public double pollardPmin1(double N, double B)
{
    double d;
    double A = 2;
    double k = 2;
    while (k<=B)
    {
        A = Math.Pow(A, k) % N;
        d = gcd(A - 1, N);
        if(1<d && d<N)
        {
            return d;
        }
        k++;
    }
    return 0;
}
```

3.4 Pollardova ρ metóda

Je faktorizačná metóda, ktorá dosahuje najlepšie výsledky v prípadoch, kedy vstup predstavuje zložené číslo s veľkým počtom malých faktorov. Metóda vyplýva z myšlienky, že nasledujúce modulárne výrazy:

$$X_0 \equiv 2 \,(\mathrm{mod}\, N) \tag{3.3}$$

$$X_{n+1} \equiv X_n^2 + 1 \,(\mathrm{mod}\, N) \tag{3.4}$$

sa v určitom momente stávajú periodickými.

Metóda je charakterizovaná nasledujúcim kódom:

```
public double pollardRho(double N)
{
    double d = 1;
    Random r = new Random();
    int rInt = r.Next(2, (int)N-1);
    double A = rInt;
    double B = rInt;
    while(d==1)
    {
        A = (Math.Pow(A, 2) - 1) % N;
        B = (Math.Pow(B, 2) - 1) % N;
        B = (Math.Pow(B, 2) - 1) % N;
        d = gcd(Math.Abs(A - B), N);
        if (d==N ||d== 1)
        {
            return 0;
        }
        else
        {
            return d;
        }
    }
    return 0;
}
```

Na výsledok ρ metódy má však značný vplyv výber počiatočných hodnôt A a B, ktoré sa typicky náhodne volia z rozsahu $\langle 2, N-1 \rangle$, čo môže mať následok v podobe zlyhania výpočtu a nutnosti opakovania riešenia s voľbou iných inicializačných parametrov A a B [9].

4 TVORBA APLIKÁCIÍ POMOCOU WCF

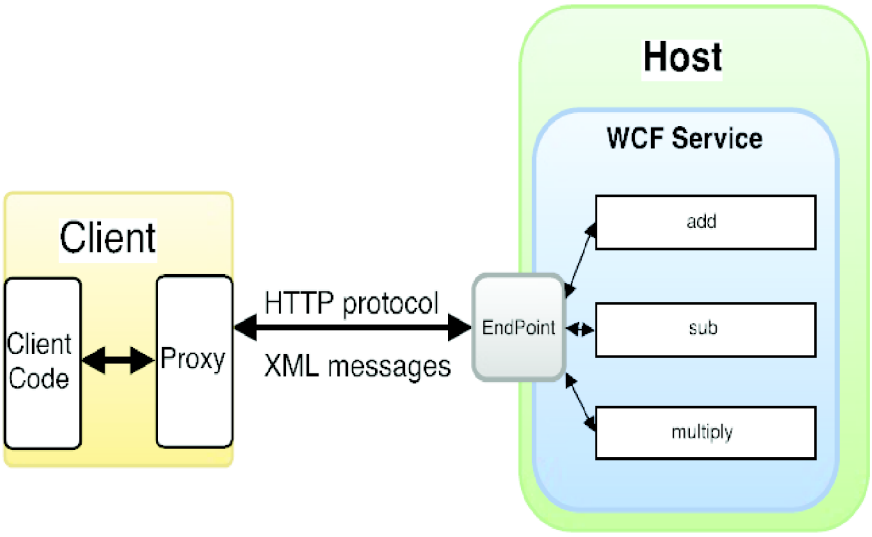
Táto kapitola pojednáva o praktickom využití WCF na niekoľkých jednoduchých príkladoch, ktoré implementujú skutočnosti uvedené v kapitole č. 2. Jednotlivé kroky tvorby tak budú detailnejšie vysvetlené. Pre vytvorenie praktických ukážok bolo použité Microsoft Visual Studio 2013, ktoré využíva platformu .NET vo verzii 4.5.

4.1 Aplikácia jednoduchej komplexnej kalkulačky

V tejto kapitole sa oboznámime so základnými prvkami tvorby WCF aplikácií. V prvej časti vytvoríme službu, ktorá pozostáva z rôznych druhov contractov, následne vytvoríme hostiteľskú konzolovú aplikáciu a v poslednej časti sa zameriame na vytvorenie konzumenta WCF služby v podobe WebKlienta.

4.1.1 Popis požiadaviek na službu

Na obr. 4.1 môžeme vidieť zjednodušený nákres požiadaviek na našu službu. Služba bude poskytovať jednoduché operácie nad komplexnými číslami, ako je sčítanie, odčítanie a násobenie. Klient, ktorý bude službu využívať, má požiadavky na spôsob komunikácie cez HTTP protokol, kde správy budú mať XML formát. V rámci príkladu tak vytvoríme endpoint, ktorý službu sprístupní klientovi.



Obr. 4.1: Schéma aplikácie komplexnej kalkulačky.

4.1.2 Tvorba WCF služby

Na začiatok si musíme vo Visual Studiu vytvoriť nový projekt, čo môžeme docieliť nasledujúcim spôsobom **File/New/Project/ClassLibrary**. Následne vložíme WCF Service, ktorý nájdeme v možnostiach **Add New Item**. Pridaním WCF služby sme docielili získanie referencie na menný priestor System.ServiceModel, ktorý je základným stavebným kameňom tvorby WCF aplikácií. Získali sme tiež implementované rozhranie, ktoré má priradený atribút [**ServiceContract**], čím dávame najavo, že sa bude jednať o WCF službu. Pod ním môžeme definovať metódy, ktoré bude služba poskytovať s atribútom [**OperationContract**]. Tu môžeme určiť vstupno-výstupné parametre jednotlivých metód využívaných klientom.

```
[ServiceContract]
public interface IComplexService
{
    [OperationContract]
    ComplexNumber add(ComplexNumber a , ComplexNumber b);
    [OperationContract]
    ComplexNumber sub(ComplexNumber a, ComplexNumber b);
    [OperationContract]
    ComplexNumber multiply(ComplexNumber a, ComplexNumber b);
}
```

Následne pristupujeme k implementácii metód definovaných v rozhraní a charakterizujeme spôsob funkcie jednotlivých operácií, v našom prípade sčítanie, odčítanie a násobenie komplexných čísel.

```
public class ComplexService : IComplexService
{
    public ComplexNumber add(ComplexNumber a, ComplexNumber b)
    {
        ComplexNumber result = new ComplexNumber();
        result.RealPart = a.RealPart + b.RealPart;
        result.ImaginaryPart = a.ImaginaryPart + b.ImaginaryPart;
        return result;
    }
}
```

Týmto je služba jednoducho vytvorená a my môžeme pristúpiť k tvorbe DataContractu pre komplexné čísla. Vytvoríme tak triedu, ktorá bude reprezentovať komplexné číslo v algebraickom tvare. Preto jej priradíme dva dátové členy typu integer. Nastavením gettru a settru zabezpečíme, že oba dátové členy budú verejne prístupné, a tým automaticky brané ako DataMember podliehajúci DataContractu, ktorým je samotná trieda komplexného čísla. Tieto atribúty teda nie je nutné uvádzať priamo v kóde. Pre podrobnejšie informácie o atribúte [**DataContract**] viď 2.5.3.

```
[DataContract]
public class ComplexNumber
{
    private int realPart;
```

```
    [DataMember]
    public int RealPart
    {
        get { return realPart; }
        set { this.realPart = value; }
    }
}
```

4.1.3 Hosťovanie WCF služby

V prípade hosťovania máme niekoľko možností, ktoré sú zhrnuté v kapitole 2.6. Pre našu službu zvolíme Self-hosting, ktorý je jednoduchý na implementáciu a poskytuje široké spektrum možností rôznych transportných protokolov. Do nami vytvoreného riešenia tak pridáme nový projekt, a to konzolovú aplikáciu, pomocou ktorej bude spustená naša služba. Aby bol klient schopný pristupovať k službe, musíme definovať endpoint, prostredníctvom ktorého bude so službou komunikovať. Na to využijeme administratívny spôsob konfigurácie endpointu pomocou **Application Configuration File**, ktorý opäť pridáme do projektu cez menu add item. Druhou možnosťou je konfigurácia programátorskou cestou s použitím inštancie ServiceHost.

V prvej časti tak za pomoci člena system.serviceModel vytvoríme službu a pomenujeme ju nasledovne „menný priestor. meno triedy“. Na základe obr. 4.1 budeme využívať jeden endpoint, ktorý nám sprostredkuje http komunikáciu . Každý endpoint musí mať pridelené tzv. **ABC (Address, Binding, Contract)**. Adresa nám určuje „Kde“ je služba umiestnená. My využijeme relatívne pomenovanie, kde hlavným dôvodom je použitie **baseAddress** v ďalšej časti **App.conf** v sekcii host nastavená na **http://localhost:9090/**. Špecifikáciou bindingu určíme, „Ako“ budeme so službou komunikovať. V prípade bindingu volíme jednu z niekoľkých možností, ktorých charakteristiky môžeme vidieť v tab. 2.1. My použijeme **basicHttpBinding**, ktorý spĺňuje požiadavky komunikácie určenej v zadaní obr. 4.1. Poslednou nutnou súčasťou endpointu je contract, ten nám určí, „Čo“ nami vytvorená služba dokáže, a tak aj operácie, ktoré nám dovolí použiť. Ako contract tak požijeme naše rozhranie, kde sú všetky tieto informácie obsiahnuté, a tým je **„ComplexService.IComplexService“**. Posledným podstatným komponentom našej služby je zabezpečenie výmeny metadát, a to môžeme docieliť dvoma spôsobmi. Prvým je povolenie **httpGetEnabled=true**, ktoré využijeme my v rámci charakteristiky spôsobu chovania **<behavior>**. Toto chovanie si pomenujeme a následne asociujeme ako parameter **behaviorConfiguration=„meno“** príslušnej služby.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
      <service name="ComplexService.ComplexService"
          behaviorConfiguration="mexBehav">
        <endpoint address="ComplexService1" binding="basicHttpBinding"
            contract="ComplexService.IComplexService"></endpoint>
        <host>
          <baseAddresses>
            <add baseAddress="http://localhost:9090/"/>
          </baseAddresses>
        </host>
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="mexBehav">
          <serviceMetadata httpGetEnabled ="true"/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

Druhou možnosťou je potom vytvorenie endpointu, ktorý bude slúžiť na výmenu metadát.

```
<endpoint address="mex" binding="mexHttpBinding"
    contract="IMetadataExchange"></endpoint>
```

Služba je tak pripravená na použitie klientom. Hosťovanie v poslednej fáze dokončíme prostredníctvom vytvorenia inštancie triedy ServiceHost predaním typu nami vytvorenej služby **(typeof)** a spustením našej služby zavolaním metódy **host.Open()**.

```
using (ServiceHost host = new ServiceHost(typeof(ComplexService.ComplexService)))
      {
          host.Open();
          Console.WriteLine("STARTED at @" + DateTime.Now.ToString());
          Console.ReadKey();
      }
```

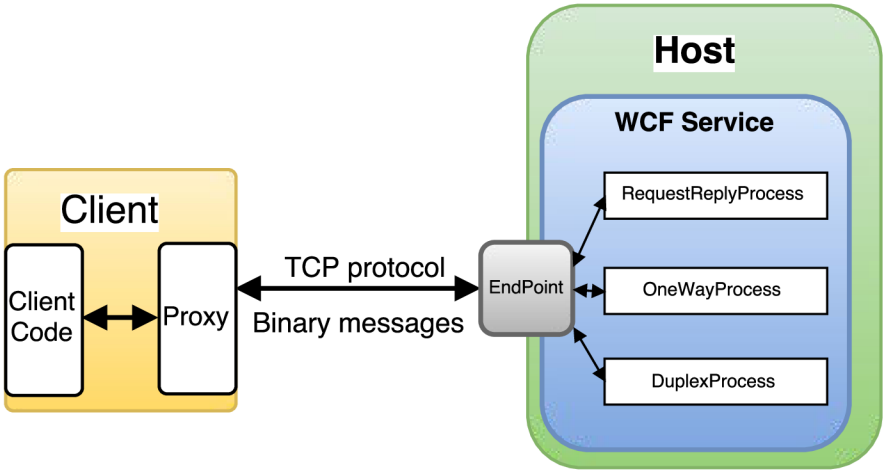
4.1.4 WCF Klient

Kód klienta, ako konzumenta nami vytvorenej služby, je možné vytvoriť zo služobných metadát, a tak použijeme buď svcutil.exe alebo ako v našom prípade **Add Service Reference**, kde jednoducho vyplníme URL služby, čím získame ServiceContract, binding a adresu endpointu. Na základe získaných informácií sme schopní vytvoriť proxy WCF klienta a prostredníctvom tejto proxy nám služba sprostredkuje operácie definované v ServiceContracte.

```
ComplexServiceRef.ComplexServiceClient client = new
ComplexServiceRef.ComplexServiceClient("BasicHttpBinding_IComplexService");
Result.Text = client.add(complexOne, complexTwo);
```

4.2 Komunikačné vzory WCF

Komunikačné vzory nám popisujú spôsob výmeny správ medzi klientom a službou. WCF poskytuje tri možnosti riešenia tejto komunikácie, a tými sú komunikácia otázka/odpoveď, jednosmerná komunikácia a duplexná komunikácia. V nasledujúcej ukážke tieto spôsoby implementujeme do aplikácie, ktorá nám ich pomôže porovnať. Na obr. 4.2 je opäť náčrt požiadaviek na našu aplikáciu. V tomto prípade však musíme použiť binding, ktorý nám duplexnú komunikáciu skutočne umožní, a preto zvolíme netTcpBinding.



Obr. 4.2: Schéma aplikácie komunikačných vzorov.

4.2.1 Tvorba WCF služby

Vytvorenie služby prebieha zhodným spôsobom ako v prvej aplikácii s nasledujúcimi úpravami pri definovaní **ServiceContractu** a jemu podriadených **OperationContractov**. V rámci služobného rozhrania definujeme tri metódy, pričom každá bude využívať iný komunikačný vzor. Prvou metódou bude RequestReplyProcess, ktorá bude sprostredkovávať rovnomenný komunikačný vzor. Komunikácia otázka-/odpoveď je defaultným vzorom, ktorý nie je potrebné špecifikovať v OperationContracte, pre názornosť však môžeme použiť vlastnosti **IsOneWay**, nastavenej na

hodnotu **false**. Druhou metódou je predstaviteľ jednosmernej komunikácie, a preto nastavujeme opäť vlastnosť **IsOneWay**, tento raz na hodnotu **true**. Zabezpečenie duplexnej komunikácie vyžaduje vytvorenie akéhosi naslúchajúceho rozhrania, na ktoré bude služba za svojho behu vysielať priebežné informácie určené klientovi. Pre našu aplikáciu to znamená vytvorenie **rozhrania pre spätné volanie** s názvom **IMessageExchangeCallback**. Toto rozhranie obsahuje metódu, ktorá bude volaná v prípade potreby, a služba tak bude oznamovať klientovi aktuálny stav behu procesu v podobe string reťazca. Dôležitým krokom je tiež prepojenie nášho primárneho rozhrania **(IMessageExchange)** s vytvoreným rozhraním na spätné volanie **(IMessageExchangeCallback)**. Toho docielime nastavením vlastnosti CallbackContract na našom primárnom rozhraní na **typ IMessageExchangeCallback**.

```
[ServiceContract(CallbackContract=typeof(IMessageExchangeCallback))]
public interface IMessageExchange
{
        [OperationContract(IsOneWay=false)]
        string RequestReplyProcess();
        [OperationContract(IsOneWay=true)]
        void OneWayProcess();
        [OperationContract(IsOneWay=false)]
        string DuplexProcess();
}
public interface IMessageExchangeCallback
{
        [OperationContract]
        void ProcessState(string state);
}
```

Opäť prevedieme implementáciu rozhrania a jednotlivých metód na hlavnú triedu služby. Pre jednoduchosť budeme v metódach používať len dočasné pozastavenie aktívneho vlákna pomocou metódy **Thread.Sleep**. Aby sme pri duplexnej komunikácii mohli priebežný stav hlásiť klientovi, využijeme metódu získania inštancie kanála, vyvolanej klientom pomocou OperationContext.Current.GetCallbackChannel<T>(), kde **T** je typ kanála, použitého na spätné volanie klienta, v našom prípade **IMessageExchangeCallback**. Duplexná komunikácia je tiež špecifická tým, že musíme umožniť prístup viacerých vlákien súčasne k našej službe, avšak pri nastavení komunikačného vzoru na otázka/odpoveď je v počiatočnom nastavení prístup viacerých vlákien zakázaný. Ako riešenie tohto problému je vhodná voľba konkurenčného **módu chovania služby** nastavená na **Reentrant**, ktorá prístup viacerých vlákien umožňuje. V nasledujúcom kóde tak môžeme vidieť definíciu metódy DuplexProcess. Obdobným spôsobom implementujeme ostatné metódy nášho rozhrania.

```
[ServiceBehavior(ConcurrencyMode=ConcurrencyMode.Reentrant)]
public class MessageExchange : IMessageExchange
{
    public string DuplexProcess()
    {
        Thread.Sleep(3000)
        OperationContext.Current.GetCallbackChannel
        <IMessageExchangeCallback>().ProcessState(DateTime.Now.ToString());
        Thread.Sleep(3000);
        return"Finished";
    }
}
```

4.2.2 Hosťovanie WCF služby

Vytvorenie hostiteľa našej služby bolo diskutované už v predchádzajúcej kap. 4.1.4, preto zhrnieme len najdôležitejšie kroky tvorby **Self-hostingu**:

1. Do nášho riešenia pridáme nový projekt **(Konzolová aplikácia)**
2. Vytvoríme referenciu na náš projekt a **System.ServiceModel**
3. Pridáme **ApplicationConfigurationFile**
4. Charakterizujeme **endpoint (A, B = netTcpBinding , C), bahavior, baseAddress**
5. Tvorba **inštancie triedy ServiceHost** s predaním typu našej služby **(typeof)**
6. Metoda **Open()**

4.2.3 WCF klient

Pre zobrazenie spôsobu chovania jednotlivých komunikačných vzorov použijeme ako klienta Windows Forms Aplikáciu. Možné riešenie GUI aplikácie je na obr. 4.3. Aby sme boli schopní vytvoriť proxy WCF klienta, musíme opäť pridať referenciu na našu službu, čo docielime vložením **URL**, definovaným v **ApplicationConfigurationFile** na hosťovskej strane. Po pridaní referencie máme možnosť vytvoriť proxy klienta a pristupovať k metódam služby. Je nutné mať na mysli, že v prípade duplexnej komunikácie je ServiCecontract úzko spätý so spätnou väzbou, ktorú sme vytvorili. Preto ak chceme využiť spätné volanie služby, musíme **rozhranie IMessageExchangeCallback**, definované v ServiceContract, implementovať aj na stranu klienta. V tomto bode určíme chovanie spätného volania na strane klienta. V nasledujúcej ukážke tak bude volanie pridávať do objektu ListOfMessages správu o priebežnom stave služby.

```
[CallbackBehavior(UseSynchronizationContext=false)]
public partial class Form1 : Form,MessageExchangeRef.IMessageExchangeCallback
{
    public void ProcessState(string state)
    {
        Control.CheckForIllegalCrossThreadCalls=false;
```

```
            ListOfMessages.Items.Add(state);
        }
}
```

Na záver priradíme udalosti jednotlivým tlačidlám GUI. Nasledujúca ukážka kódu definuje spôsob chovania tlačidla Duplex. Riešenie udalostí ostatných tlačidiel bude obdobné. Najskôr vytvárame proxy klienta. Jeho konštruktor však vyžaduje ako vstupný parameter inštanciu triedy, ktorá implementuje u klienta rozhranie určené na spätné volanie. Inštanciu našej triedy (Form1) tak získavame prostredníctvom vytvorenia objektu triedy **InstanceContext (System.ServiceModel)**. Objekt tak môžeme predložiť nášmu proxy konštruktorovi, čím získame prístup k metódam služby. Rovnako, ako pri tvorbe služby, musíme riešiť problematiku chovania vlákna, na ktorom beží naše užívateľské rozhranie. Základné vlákno užívateľského rozhrania totiž beží v móde, kedy v prípade žiadosti o službu nie je schopné priebežne reagovať na spätné volanie. Preto musíme zabezpečiť, aby bolo vytvorené nové nezávislé vlákno, a to docielime definíciou chovania spätného volania na **„UseSynchronizationContext=False“**.

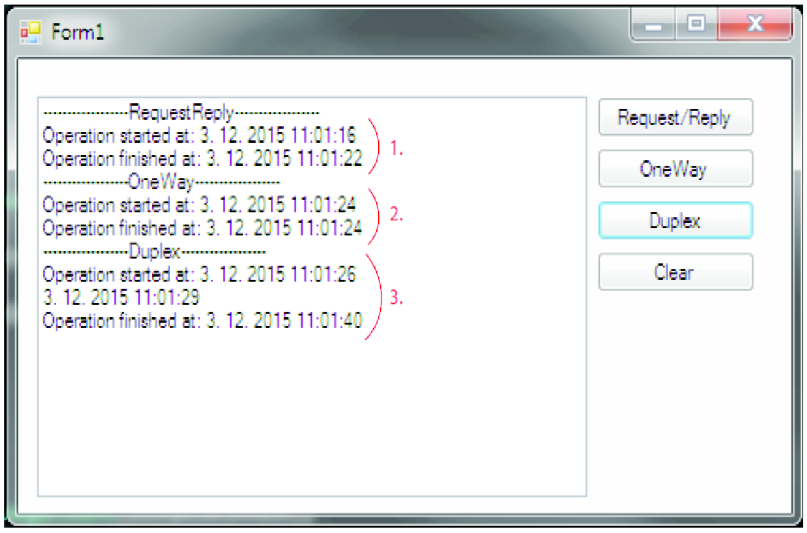
```
[CallbackBehavior(UseSynchronizationContext=false)]
public partial class Form1 : Form,MessageExchangeRef.IMessageExchangeCallback
{
    private void DuplexButton_Click(object sender, EventArgs e)
    {
        InstanceContext instanceContext= new InstanceContext(this);
        MessageExchangeRef.MessageExchangeClient client =new
            MessageExchangeRef.MessageExchangeClient(instanceContext);
        ListOfMessages.Items.Add(String.Format("Operation started at: {0}",
            DateTime.Now.ToString()));
        string result =client.DuplexProcess();
        ListOfMessages.Items.Add(String.Format("Operation finished at: {0}",
            DateTime.Now.ToString()));
    }
}
```

4.2.4 Výstupy aplikácie

Na základe obr. 4.3 môžeme porovnať princíp chovania jednotlivých komunikačných vzorov, ktoré WCF poskytuje:

1. Komunikácia otázka/odpoveď: Klient posiela žiadosť na našu službu a následne čaká na odpoveď. V priebehu čakania nemôže pokračovať vo vykonávaní svojho procesu až do bodu, kedy príjme odpoveď ako výsledok procesu služby. Toto chovanie je jasné z výpisov GUI, kde klient spracuje žiadosti o výpis typu komunikácie a počiatočného času a následne zažiada o službu. Chod služby sprostredkuje uspanie vlákna na dobu šesť sekúnd, po ktorej klient získa odpoveď a následne pokračuje vo svojej činnosti.

2. Jednosmerná komunikácia: Predstavuje komunikáciu pri použití jedinej správy, kedy klient zavolá našu službu a okamžite pokračuje vo vykonávaní svojho procesu. Služba vykoná proces, ktorý sme jej určili, avšak na stranu klienta už neodpovedá.
3. Duplexná komunikácia: Zabezpečuje možnosť získavania informácií zo spätných volaní v priebehu riešenia otázky, zaslanej klientom na stranu služby. Na výpise tak môžeme vidieť priebežný výpis klienta, ktorý je vygenerovaný v priebehu čakania na odpoveď.



Obr. 4.3: Výstupy aplikácie komunikačných vzorov.

4.3 Inštančné módy služby

V momente, kedy klient vyšle žiadosť na stranu služby, dochádza k vytvoreniu inštancie triedy našej služby. Životnosť vytvorenej inštancie môže byť rôzne dlhá – od inštancií, ktoré sú vytvorené vždy len na dobu jedného volania metódy služby, až po inštancie pokrývajúce celý životný cyklus služby. Inštančné módy tak diktujú spôsob vytvárania inštancií služby po prijatí žiadosti od klienta. WCF nám poskytuje tri základné inštančné módy, ktoré si predvedieme na jednoduchom príklade:

1. PerCall
2. PerSession
3. Single

4.3.1 PerCall

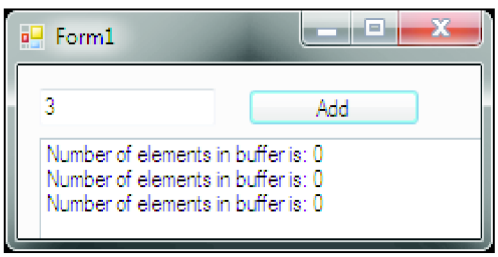
Je mód, pri ktorom každá žiadosť zo strany klienta spôsobí vznik novej inštancie služby, pričom nezáleží na tom, či sa jedná o zhodného klienta alebo úplne odlišného. Z toho vyplýva, že po naplnení vyžiadaných operácií a odpovedi klientovi dochádza k okamžitému zániku inštancie služby. Nasledujúci príklad je vytvorený na základe obdobného postupu, ako služby v predchádzajúcich kapitolách, až na úpravy, ktoré reprezentuje nasledujúci kód z definície služby. Aby sme boli schopní určiť inštančný mód našej službe, musíme charakterizovať spôsob chovania služby, kde nastavíme hodnotu **InstanceContextMode na InstanceContextMode.PerCall**.

```
[ServiceBehavior(InstanceContextMode=InstanceContextMode.PerCall)]
public class BufferService : IBufferService
{
    private List<int>Buffer =new List<int>();
    public void AddElement(int Number)
    {
        Buffer.Add(Number);
    }
    public int NumOfElement()
    {
        returnBuffer.Count;
    }
}
```

Klientska aplikácia využíva len jednoduché zobrazenie v podobe ListBoxu a tlačidla na pridanie čísla do zásobníka.

```
BufferServiceRef.BufferServiceClient client=new BufferServiceRef
    .BufferServiceClient("NetTcpBinding_IBufferService");
client.AddElement(Int16.Parse(NumberBox.Text));
client.AddElement(Int16.Parse(NumberBox.Text));
client.AddElement(Int16.Parse(NumberBox.Text));
ListCommand.Items.Add(string.Format("Number of elements in buffer
    is:{0}",client.NumOfElement()));
```

Z výstupov na obr. 4.4 je pochopiteľné, že každé volanie metódy našej služby má za následok vznik novej inštancie WCF služby. Táto skutočnosť vyplýva z hodnoty, ktorá udáva veľkosť zásobníka našej služby. Tá je aj po pridaní troch čísel rovná nule.



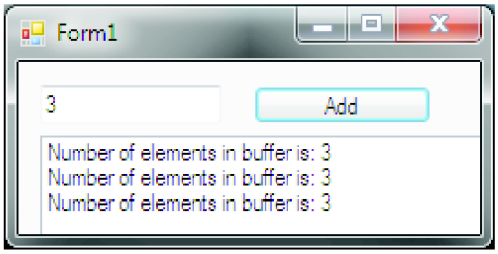
Obr. 4.4: Výstup aplikácie po pridaní troch hodnôt do zásobníka pri použití PerCall módu.

4.3.2 PerSession

Je mód, pri ktorom dochádza k vytvoreniu inštancie služby pri vzniku novej relácie medzi klientom a službou. Každý pripojený klient využíva svoju inštanciu služby a môže tak volať metódy služby bez jej zániku. Inštancia je spravovaná až do zániku relácie. Implementácia PerSession módu je možná nasledujúcou úpravou nášho kódu.

```
[ServiceBehavior(InstanceContextMode=InstanceContextMode.PerSession)]
public class BufferService : IBufferService
{
...
}
```

Obrázok 4.5 dokazuje, že pri spustení udalosti tlačidla našej aplikácie dôjde k vytvoreniu proxy klienta, a tým aj k vzniku inštancie služby. Následne používame volanie niekoľkých metód pridania prvku a vo finálnej fáze výpis počtu prvkov. Počet prvkov zásobníka sa vyšplhal na hodnotu tri, čo súhlasí s počtom volaní pridania prvku a vidíme tak, že všetky volania prebehli v rámci jednej inštancie služby. Opätovným spustením udalosti tlačidla dosahujeme zhodných výsledkov. Dôvodom je nový proxy klient a s ním spojená inštancia služby.



Obr. 4.5: Výstup aplikácie po pridaní troch hodnôt do zásobníka pri použití PerSession módu.

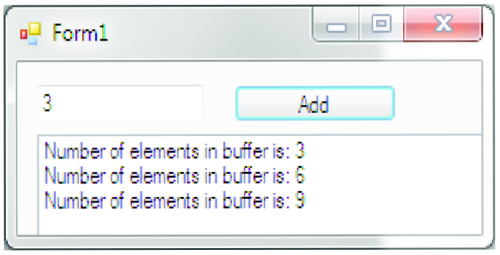
4.3.3 Single

Inštancia služby je u Single módu vytvorená len raz za celý životný cyklus služby. Táto inštancia má za úlohu uspokojovať všetky volania klientov, pričom nezáleží na ich počte. Mód tak zaručí globálne využívanie prostriedkov medzi klientmi. Single mód definujeme nasledujúcim spôsobom:

```
[ServiceBehavior(InstanceContextMode=InstanceContextMode.Single)]
public class BufferService : IBufferService
{
...
}
```

Na obr. 4.6 je výpis nášho programu, ktorý dokazuje, že pri single móde vzniká len jedna inštancia služby. Preto aj v prípade, keď prebehne viac relácií medzi klientom

a službou, nedochádza k zmazaniu nášho zásobníka, ale inverzne k jeho zvyšovaniu. Zásobník sa tak stáva globálnou premennou spoločne využívanou všetkými pripojenými klientmi. Zmazanie inštancie prebehne až v momente, kedy dôjde k ukončeniu našej služby a opätovnému spusteniu.



Obr. 4.6: Výstup aplikácie po pridaní troch hodnôt do zásobníka pri použití Single módu.

5 NÁVRH DISTRIBUOVANEJ APLIKÁCIE NA FAKTORIZÁCIU ČÍSEL

Kapitola je zameraná na detailný popis návrhu a následnej implementácie distribuovanej aplikácie, ktorej primárnou úlohou bude rozklad zložených čísel na prvočísla. Sústredíme sa na spôsob riešenia základných WCF komponentov ako služby realizácie rozkladu na prvočísla klienta, ktorého hlavnou úlohou je riadenie staníc, ktoré predstavujú výpočtovú silu distribuovanej aplikácie. Pre riešenie problému faktorizácie zvolíme jednu z metód uvedených v kapitole 3, ktorá bude vhodne modifikovaná, aby výpočet bolo možné rozdeliť medzi väčšie množstvo prihlásených zariadení. Dôležitou súčasťou tejto kapitoly je riešenie spôsobu komunikácie a kooperácie jednotlivých komponentov. Na záver bude porovnaný výkon mnou navrhnutej distribuovanej aplikácie s aplikáciou, ktorá bude využívať zhodnú metódu rozkladu, avšak jej funkcionalita je obmedzená na jediné zariadenie. Pre samotnú realizáciu programu budeme využívať VisualStudio 2013 s použitím platformy .NET verzie 4.5. podobne ako v kapitole 4.

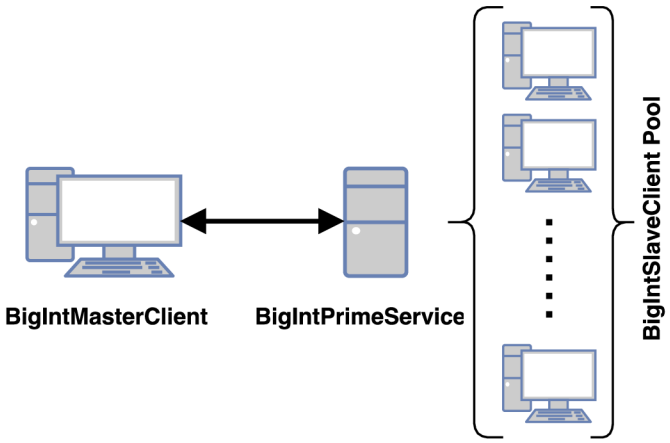
5.1 Voľba vhodnej metódy faktorizácie čísel

V teoretickej časti 3 sme si predstavili základných reprezentantov faktorizácie čísel, kde pre každú metódu môžeme hovoriť o rôznych plusoch a mínusoch. Z hľadiska distribúcie problému na viacero zariadení sa javí ako veľmi vhodná metóda postupného delenia, kde môžeme prehľadávaný interval $\langle 2, \sqrt{N} \rangle$ rovnomerne rozdeliť medzi ľubovoľný počet zariadení, a tým urýchliť získanie faktorov zloženého čísla. Metódu môžeme drobnými úpravami zrýchliť najmä prostredníctvom sita k eliminácii hodnôt, ktorých jedným z faktorov je číslo 2. V tomto prípade je tiež dôležité, že v bode, kedy špecifikovaná zložená hodnota neobsahuje faktor v danom intervale, označujeme ju za prvočíslo.

5.2 Štruktúra distribuovanej aplikácie

Vytvorenie distribuovanej aplikácie predstavuje komplexný problém pozostávajúci z viacerých prvkov, ktoré medzi sebou vhodným spôsobom komunikujú bez toho, aby sa narušil ich samostatný beh. Na obr. 5.1 môžeme vidieť schému aplikácie určenej k faktorizácii čísel. Obrázok odhaľuje tri dôležité prvky, s ktorými budeme pracovať. Prvým nosným prvkom je služba BigIntPrimeService. Tá je centrálnym bodom celej distribuovanej aplikácie. Jedine pomocou tejto služby sme schopní zabezpečiť kooperáciu viacerých zariadení na konkrétnom úkone. Ďalším podstatným elementom je

klientská aplikácia určená k riadeniu distribuovaného výpočtu BigIntMasterClient. Tohto klienta využívame ku komunikácii so samotným užívateľom aplikácie, pričom klient do faktorizačného procesu neprispieva vlastnými výpočtami. Užívateľ tak môže špecifikovať hodnotu ku faktorizácii, voliť rôzne módy faktorizácie a ich sekundárne nastavenia. Z hľadiska realizácie tohto klienta bola zvolená Windows Forms aplikácia, a to najmä z dôvodu širokých možností riešenia GUI, kde môžeme využiť veľké množstvo vstavaných objektov, ktoré sú prehľadné, jednoduché na nastavenie a priradenie vhodných funkcií. Poslednou súčasťou distribuovanej aplikácie je výkonný klient BigIntSlaveClient. Tento klient je aktivovaný na požadovanom počte zariadení, ktoré využívajú funkcie navrhnuté k faktorizácii zloženého čísla určeného BigIntMasterClientom. Klienti sa tak spoločnými výpočtami podieľajú na finálnom výsledku distribuovaného výpočtu, reprezentovaného na záver riadiacemu klientovi.

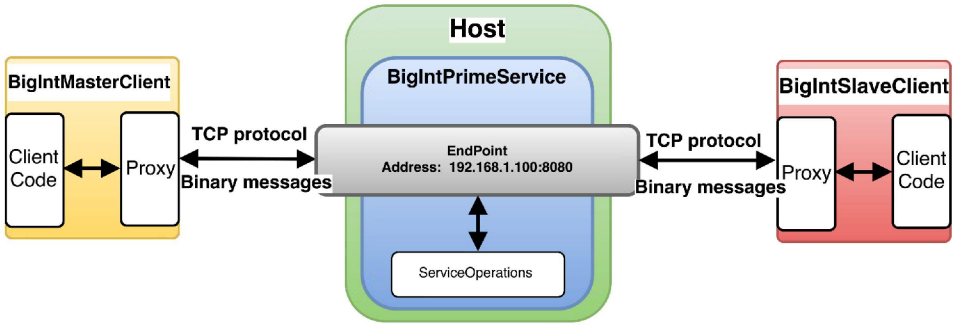


Obr. 5.1: Architektúra distribuovanej aplikácie.

5.3 Spôsob komunikácie jednotlivých prvkov distribuovanej aplikácie

Zabezpečenie komunikácie prvkov aplikácie je kľúčovým bodom návrhu. Celé riešenie je tak postavené na použití technológie WCF. Špecifikácia tejto technológie a jej možné použitie boli uvedené v kapitole 2. Centrálnym prvkom aplikácie je WCF služba, s ktorou je nadväzovaný kontakt zo strany riadiaceho klienta i výpočtového klienta. Na obr. 5.2 môžeme vidieť, že ku komunikácii so službou sa využíva netTcpBinding, ktorý bližšie definuje spôsob chovania a kódovania endpointu služby. Tento druh bindingu bol zvolený hlavne z dôvodu možnosti využitia duplexnej komunikácie, bezpečnosti a podpory inštančných módov. Z obrázku je tiež zreteľné, že

služba po aktivácii funguje na jej priradenej privátnej IP adrese, v našom prípade 192.168.1.100 na porte 8080. Celú konfiguráciu endpointu a chovania komunikácie medzi prvkami aplikujeme na súbor App.config. Ten je vo formáte XML obdobne ako v prípade 4.1.3. Každý klient následne pridáva referenciu na túto službu a jej endpoint. Po získaní referencie sa tak jednoducho vytvorí klientska časť. Tá je následne využívaná k vytvoreniu proxy a prostredníctvom nej sme schopní smerovať požiadavky na služobný endpoint.



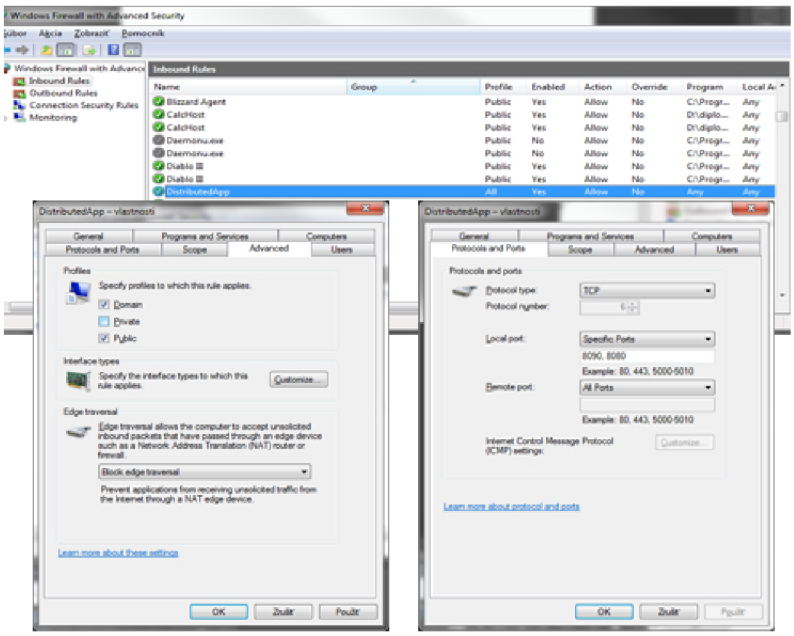
Obr. 5.2: Schéma prvkov distribuovanej aplikácie.

Pri reálnom nasadení služby do prevádzky je nutné dbať na niekoľko opatrení, bez ktorých by klienti neboli schopní k službe na lokálnej sieti pristúpiť. V prvom rade je nutné hosťovskú aplikáciu služby aktivovať na zariadení s administrátorskými právami. Ďalšou možnou požiadavkou je rezervovanie konkrétnej URL, ktorú služba využíva k možnému prístupu užívateľov a účtov bez administrátorského oprávnenia. Toho môžeme docieliť využitím nasledujúceho príkazu v príkazovom riadku:

$$\text{netsh http addurlacl url} = \text{http} : //192.168.1.100 : 8080/\text{user} = \backslash\text{everyone} \quad (5.1)$$

Posledným krokom je úprava pravidiel Windows Firewallu, aby nebola odfiltrovaná a následne zahodená komunikácia smerovaná na port 8080. V pokročilých nastaveniach firewallu nastavíme pravidlo na konkrétny port, pre ktorý povolíme prichádzajúce data (Inbound Rule). V definícii pravidla uvádzame platnosť pre všetky možné profily. Tento postup je znázornený na obrázku 5.3.

Komunikácia zo strany služby ku klientskym aplikáciám využíva skutočnosť, že netTcpBinding podporuje duplexnú komunikáciu. V tomto prípade využívame modifikovaný Publisher/Subscriber model, založený na WCF službe. K tejto službe sa hlásia odberatelia (BigIntSlaveClient) informácií, ktorých zdrojom je BigIntMasterClient. Služba beží v inštančnom móde Single. Single mód bol podrobnejšie priblížený v kapitole 4.3.3. Jediná inštancia služby, trvajúca celý životný cyklus, nám



Obr. 5.3: Postup nastavenia poročilého pravidla pre Windows Firewall.

umožňuje vytvorenie generického slovníka rozhraní určených na spätné volanie prihláseným odberateľom, kde ku každému rozhraniu priraďujeme informáciu o IP adrese klienta, zvoleného pre komunikáciu. Služba v každom momente jej behu dokáže pristupovať ku všetkým prihláseným klientom, kde jediným spôsobom, ako ukončiť spoluprácu s ľubovoľným klientom, je jeho vypnutie a s tým spojené odhlásenie od služby.

5.4 Riešenie GUI jednotlivých prvkov aplikácie

V tejto časti sa oboznámime s vizuálnou formou jednotlivých prvkov aplikácie a funkcionalitami, ktoré zabezpečujú. Obrázok 5.4 nám ukazuje tri hlavné prvky aplikácie, s ktorými užívateľ prichádza do kontaktu.

Centrálna časť obrázku znázorňuje riešenie WCF služby, a to konkrétne hostiteľskú entitu, založenú na použití jednoduchej konzolovej aplikácie. Konzolové riešenie nám umožňuje využiť iba jednoduché výpisy označujúce aktuálny stav, prípadne informácie o klientoch, ktorí sa počas behu služby prihlásili k odberu, alebo ho zrušili.

GUI spojené s riadiacim klientom je zobrazené v ľavej časti obrázka, jedná sa teda o Windows Forms aplikáciu. K dispozícii tak máme rozšírené možnosti zobrazovania v porovnaní s hostiteľom. Prevažnú časť aplikácie tvoria dva ListBox prvky, kde prvok v ľavej časti využívame na zobrazenie aktuálne pripojených výpočtových klientov, a to pomocou ich IP adries. Tento ListBox je previazaný s niekoľkými

funkciami zabezpečujúcimi aktuálnosť zoznamu ako na strane riadiaceho klienta, tak služby. Adresy zobrazené v zozname teda patria klientom, ktorí sú pripravení podieľať sa faktorizácii zvoleného čísla. Druhý ListBox je určený k zobrazovaniu primárnych informácií spojených s chodom aplikácie a výsledkami faktorizácie. Zvyšok aplikácie tvoria riadiace objekty Button, TextBox a RadioButton prvky. Máme teda dve základné možnosti pri realizácii faktorizácie. V prípade, že užívateľ zvolí „Classic“ mód výpočtu, tak má jedinú možnosť, a tou je špecifikácia zloženej hodnoty určenej k faktorizácii a následné potvrdenie zadania výpočtu SlaveKlientom prostredníctvom tlačidla PrimeFactorization. Druhý modifikovaný režim „Modified“ využíva k svojim výpočtom vopred vygenerovaný generický list prvočísel, čím urýchľuje výpočet ďalšou redukciou prehľadávanej oblasti jednotlivým výpočtovým klientom. List je následne uchovávaný počas celého behu SlaveClientForm aplikácie a môže byť viacnásobne využívaný. Po zvolení režimu je nutné definovať akýsi limit tohto listu prvočísel. Tento limit je možné potvrdiť prostredníctvom Initialize tlačidla a až následne môžeme určiť hodnotu k faktorizácii. Inicializácia je podľa voľby limitnej hodnoty značne časovo náročná, a je preto dôležité s voľbou zloženého čísla počkať až do momentu, kedy budú všetky SlaveClientForm pripravené k výpočtu (režim „Ready“). Následne je možné opäť pristúpiť k samotnej faktorizácii (tlačidlo „PrimeFactorization“). Pri voľbe zloženej hodnoty však musíme byť opatrní. Hodnota nesmie prekročiť hranicu limitu zvoleného pri inicializácii, alebo musí byť zložená z faktorov spadajúcich do určeného limitu.

Posledným prvkom distribuovanej aplikácie je BigIntSlaveClient, ktorý bol vytvorený tiež ako Windows Forms aplikácia. Klientská aplikácia je riešená jednoduchým spôsobom. Môže sa nachádzať v štyroch základných módoch, o ktorých užívateľa informuje. Tými módmi sú:

- Listening: klient sa úspešne prihlásil ako odberateľ našej služby a je pripravený prispievať k výpočtu faktorov („Classic“ mód)
- Ready: klient informuje užívateľa o dokončení inicializácie listu prvočísel („Modified“ mód)
- Counting: klient v danom momente aplikuje faktorizačný algoritmus na výpočet zobrazovanej hodnoty
- Finished: klient výpočet ukončil a informuje o tom službu

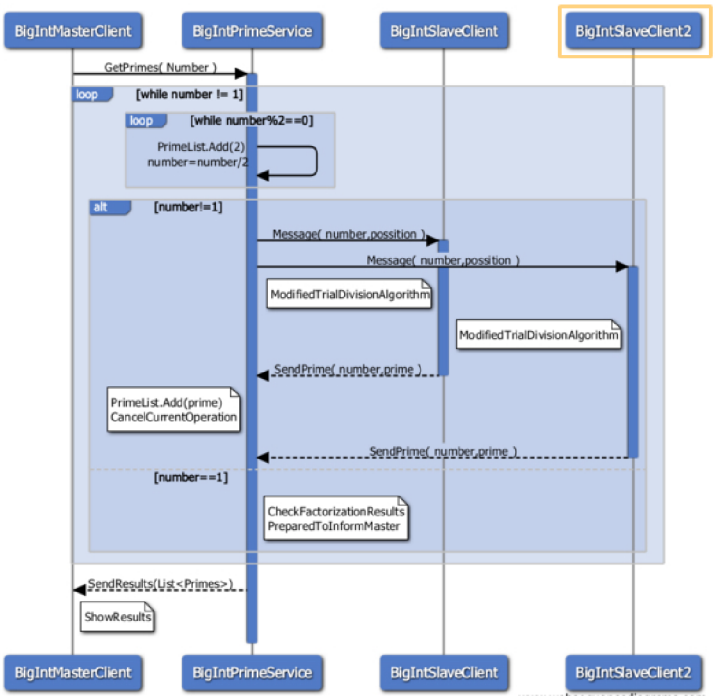
5.5 Metodika funkcií navrhnutých pre riešenie problému faktorizácie

Na základe teoretických poznatkov z predchádzajúcich kapitol, ktoré som využil k návrhu aplikácie s distribuovaným výpočtom prvočíselného rozkladu zloženého

Obr. 5.4: Predstaveie GUI prvkov aplikácie.

čísla, môžeme funkciu aplikácie zhrnúť do nasledujúceho zjednodušeného sekvenčného diagramu viď obr. 5.5. Prvotný podnet je teda riešený zo strany BigIntMasterClienta, využívajúceho volanie metódy GetPrimes so vstupným parametrom v podobe hodnoty, ktorú chceme rozložiť na prvočísla. Túto metódu sme na strane služby definovali v rámci ServiceContractu, preto je k dispozícii všetkým odberateľom našej BigIntPrimeService. Z obrázku je tiež zrejmé, že služba beží celú dobu po aktivácii v jedinej inštancii. V rámci definície služby charakterizujeme chovanie našej služby na **InstanceContextMode.Single** obdobne ako v kapitole 4.3.3. Služba ako taká musí tiež zvládať obstarávať viacero vlákien, ktoré v danom momente môžu službu volať a z toho dôvodu musíme nastaviť konkurenčný mód služby na Multiple. Tento spôsob chovania definujeme v časti **[ServiceBehavior(ConcurrencyMode =ConcurrencyMode.Multiple)]**. Metóda GetPrimes obsahuje niekoľko rozhodovacích mechanizmov. Tieto mechanizmy si priblížime v nasledujúcej časti. V prvom rade je dôležité, že služba obsahuje niekoľko privátnych premenných, do ktorých zaznamenáva priebežné informácie využívané k fungovaniu služby, ako napr. nájdené faktory definovaného zloženého čísla, rozhrania pre spätné volanie a flagy, ktoré predstavujú akési rozcestia pri behu služby. Hlavným prvkom je generický list prvočísel, ktorý je využívaný v každej iterácii bežiacej služby na zaznamenávanie jednotlivých faktorov nájdených buď samotnou službou alebo za pomoci BigIntSlaveClientov. Prvá fáza behu služby je založená na aplikovaní sita pre odfiltrovanie všetkých násobkov faktoru 2, kde výpočet v tejto chvíli prebieha bez distribúcie z dôvodu efektivity výpočtu. Po odfiltrovaní všetkých násobkov 2 služba pokračuje rozdistribuovaním zvyškovej hodnoty medzi prihlásených odberateľov určených na realizáciu výpočtu. Na informovanie BigIntSlaveClientov využije služba kanál určený na spätné volanie, kde pomocou metódy Message zasiela údaje o zvyškovej hodnote a pozícii, umožňujúcej klientovi získať interval, v ktorom bude hľadať faktory daného čísla. Po získaní

intervalu, určeného k prehľadávaniu, klienti využívajú najmä operáciu modulo k zisteniu, či je vstupná hodnota deliteľná bez zvyšku všetkými nepárnymi číslami z rozsahu. V prípade, že BigIntSlaveClient nájde faktor zloženého čísla, tak o tom okamžite informuje službu, ktorá si faktor zaznamená a informuje všetkých ostatných klientov o získaní faktora, čo má za následok ukončenie faktorizačného algoritmu pre pôvodne zadanú hodnotu. Podobne ako v prípade služby, ak má klient reagovať na viacnásobné prichádzajúce spätné volania, je opäť podstatné nastaviť spôsob chovania týchto volaní použitím [**CallbackBehavior(ConcurrencyMode= ConcurrencyMode.Multiple)**]. Vzájomná komunikácia a operácie výpočtových klientov a služby prebiehajú naďalej až do momentu, kedy služba na základe vyhodnotenia vstupnej hodnoty podľa výroku, že vstupná hodnota je rovná 1, usúdi, že došlo k naplneniu požadovanej funkcie našej služby. Tým sa obsah generického listu faktorov stáva výsledkom našej faktorizácie a po overení je pripravený na zaslanie riadiacemu klientovi (BigIntMasterClient). Klient následne pristúpi k zobrazeniu výsledkov užívateľovi.



Obr. 5.5: Zjednodušený sekvenčný diagram aplikácie distribuovaného výpočtu faktorizácie čísel.

5.6 Popis vybraných metód distribuovanej aplikácie

Kapitola je venovaná popisu jednotlivých metód využívaných distribuovanou aplikáciou s použitím konkrétnych ukážok kódu. Najskôr si priblížime niekoľko dôležitých metód a úryvkov kódu, ktoré sú využívané službou k sprostredkovaniu možnosti faktorizácie čísel. Z hľadiska definície a sprístupnenia našej WCF služby s názvom BigIntPrimeService pristupujeme k špecifikácii ServiceContractu, ktorý pozostáva z niekoľkých OperationContractov. Tie nám určujú metódy, ktoré bude možné volať zo strany klientov.

V ukážke môžeme vidieť definíciu rozhrania IBigIntPrimeService, kde v prvej časti určujeme dôležité vlastnosti, ktorými sú nastavenie SessionMode a CallbackContract. Na základe nastavenia SessionMode na hodnotu Required docielime to, že služba bude podporovať len také spôsoby komunikácie s endpointom, ktoré umožňujú vytváranie relácií. Preto naša aplikácia využíva ku komunikácii netTcpBinding. V opačnom prípade by bola služba okamžite ukončená výnimkou. V rámci druhej vlastnosti priraďujeme hlavnému rozhraniu našej služby ešte podriadené rozhranie IBigIntPrimeCallback, ktoré obsahuje metódy slúžiace na spätné volanie klientom. Vytvorené rozhrania následne obsahujú metódy s rôznymi vstupnými parametrami, ktoré slúžia na komunikáciu medzi účastníkmi faktorizácie. Každej metóde sme nastavili vlastnosť IsOneWay na hodnotu true, ktorá nám umožní vyslať zo strany klienta požiadavku na službu bez toho, aby pôvodca správy očakával odpoveď.

```
[ServiceContract(SessionMode = SessionMode.Required, CallbackContract =
    typeof(IBigIntPrimeCallback))]
public interface IBigIntPrimeService
{
    [OperationContract(IsOneWay = true, IsInitiating = true)]
    void SlaveSubscribe(string ipAddress);
    [OperationContract(IsOneWay = true)]
    void GetPrimes(BigInteger num,BigInteger prime,bool flag,int mode);
    ...
}
public interface IBigIntPrimeCallback
{
    [OperationContract(IsOneWay = true)]
    void Message(int position,int subs,BigInteger number,int mode);
    ...
}
```

Definované rozhrania sú ďalej implementované do hlavnej triedy našej služby. Úvodná časť opäť určuje chovanie našej služby, v tomto prípade je to vlastnosť InstanceContextMode nastavená na Single, ktorá určuje, že služba pobeží v jednej inštancii po celú dobu jej života. Druhou vlastnosťou je ConcurrencyMode určený ako Multiple, ktorého úlohou je zabezpečiť schopnosť služby reagovať v danom mo-

mente na väčšie množstvo žiadostí klientov.

```
[ServiceBehavior(InstanceContextMode = InstanceContextMode.Single, ConcurrencyMode
    = ConcurrencyMode.Multiple)]
public class BigIntPrimeService : IBigIntPrimeService
{
    ...
}
```

Po implementácii rozhrania je podstatným prvkom definícia všetkých metód, ktoré patria pod základné rozhranie služby IBigIntPrimeService. Ukážka zobrazuje spôsob implementácie metód SlaveSubscribe a časť metódy GetPrimes, ktorá je zodpovedná za rozoslanie žiadosti o riešenie faktorizácie zloženého čísla medzi prihlásených výpočtových klientov. Metóda SlaveSubscribe slúži na prihlásenie klienta k odberu našej služby. Ako vstupný parameter je použitý string reťazec, ktorý pozostáva z informácií, ako je IP adresa a SessionId prebiehajúcej relácie. Následne získavame inštanciu aktuálneho kanála pre spätné volanie a overujeme aktuálnosť slovníka našich odberateľov. Ak inštancia kanála spätného volania existuje, ukončíme prihlasovanie odberateľa. V opačnom prípade pristupujeme k vytvoreniu nového záznamu odberateľa pomocou metódy pridania dvojice kľúča a hodnoty (inštancia spätného volania, IP adresa) do vopred vytvoreného slovníka. Ďalším krokom je využitie metódy na zasielanie zoznamu hodnôt nášho slovníka smerom k prihláseným riadiacim klientom. Posledným krokom je výpis krátkej informačnej správy o prihlásení odberateľa do konzoly služby.

```
public void SlaveSubscribe(string ipAddress)
{
    IBigIntPrimeCallback callback =
        OperationContext.Current.GetCallbackChannel<IBigIntPrimeCallback>();
    if (!SlaveSubscribers.ContainsKey(callback))
    {
        SlaveSubscribers.Add(callback, ipAddress);
        foreach(IBigIntPrimeCallback call in MasterSubscriber)
        {
            call.SendSubscriberList(SlaveSubscribers.Values.ToList());
        }
        Console.WriteLine(DateTime.Now.ToString() + ": SlaveClient:" + ipAddress +
            " has joined SlavePool");
    }
}
```

GetPrimes metóda je ústrednou časťou celej WCF služby a my si predstavíme krátky úryvok spojený s distribuovaním problému smerom ku klientom. Z hľadiska možnosti využitia tejto metódy na strane riadiaceho klienta i na výpočtovej strane sme pristúpili k využitiu flag premennej v kóde. Táto premenná tvorí rozcestník v behu služby, kedy v prípade, že je hodnota flag nastavená na true, tak služba vyhodnotí, že sa jedná o odpoveď od výpočtového klienta. Odpoveď potom obsahuje údaj o nájdenom prvočísle, a tiež upravenú hodnotu zloženého čísla, ktoré bude ďalej faktorizované.

Získané prvočíslo následne pridávame do výsledného generického listu pomocou metódy Add. V ďalšom kroku pristupujeme k rozdistribuovaniu informácie o ukončení aktuálne prebiehajúceho, už dokončeného výpočtu (ReportCancel) a pokračujeme vyslaním nového modifikovaného zloženého čísla na všetkých pripojených výpočtových klientov (Message). Každý klient získa údaje o pozícii a počte odberateľov, ktoré využije k výpočtu intervalu určeného na faktorizáciu zloženého čísla. Informácia o samotnom zloženom čísle je veľmi dôležitou súčasťou správy. Posledným vstupným parametrom je definícia módu, ktorý sa používa k rozlíšeniu možností výpočtu faktorizácie a je určený riadiacim klientom (Classic, Modified).

```
public void GetPrimes(BigInteger num, BigInteger prime, bool flag, int mode)
{
    ...
    else
    {
        if (flag)
        {
            Result.Add(prime);
        }
        foreach (IBigIntPrimeCallback call in SlaveSubscribers.Keys)
        {
            call.ReportCancel();
            call.Message(counter, SlaveSubscribers.Count, num, mode);
            counter++;
        }
    }
    ...
}
```

Spôsob spracovania správy zaslanej službou výpočtovým klientom môžeme vidieť v nasledujúcej časti kódu. V prvom bode vytvárame proxy nášho klienta, a tak sprístupníme metódy poskytované službou. Aby bolo možné počas behu metódy pre spätné volanie pristupovať ku grafickým objektom, je nutné vypnúť detekciu krížových dotazov na jednotlivé vlákna. Následne rozlišujeme zvolený mód. V našom prípade bol zvolený „Classic“. V tejto časti je opäť využívaný boolean výraz určený k rozlíšeniu ukončenia výpočtu prvočísla. Prostredníctvom tejto hodnoty sme schopní ukončiť algoritmus faktorizácie (getPrimeNumber) aj za jeho behu, a to v momente, kedy nás o tom informuje služba na základe nájdenia faktoru iným zariadením. Faktorizačný algoritmus nám potom produkuje dvojicu hodnôt (nové zložené číslo, faktor), ktoré reprezentujú výsledok faktorizácie. V prípade, že je výsledkom hodnota nula, môžeme tvrdiť, že v príslušnom intervale sa nenachádza faktor vstupného čísla. Predpokladáme, že faktor aktuálnej hodnoty bude nájdený iným zariadením. Keď je v tomto bode faktor nájdený, pristupujeme k informovaniu služby, ktorá rozhodne o nasledujúcom postupe.

```
public void Message(int position, int subs, BigInteger number, int mode)
{
    InstanceContext instanceContext = new InstanceContext(this);
    BigIntPrimeService.BigIntPrimeServiceClient client = new
        BigIntPrimeService.BigIntPrimeServiceClient(instanceContext);
    Control.CheckForIllegalCrossThreadCalls = false;
    label1.Text = "Counting " + number.ToString();
    KeyValuePair<BigInteger, BigInteger> result = new KeyValuePair<BigInteger,
        BigInteger>();
    jobFinished = false;
    result = supportFunc.getPrimeNumberClassic(position, subs, number);
    if (result.Key != 0)
    {
        try
                                {
            client.GetPrimes(result.Key, result.Value, true, mode);
        }
        catch (EndpointNotFoundException ex)
        {
            label1.Text = "Service OFFLINE";
        }
    }
    label1.Text = "Finished";
}
```

V momente, keď služba vyhodnotí, že výpočet bol úspešne ukončený, dochádza k informovaniu riadiaceho klienta, ktorému zasiela list faktorov zvoleného čísla. Riadiaci klient má po prijatí výsledkov za úlohu riešiť interakcie s užívateľom, a to prostredníctvom vhodného výpisu. Táto časť obsahuje aj implementáciu triedy Stopwatch, určenej na meranie presného času behu aplikácie, kde odštartovanie stopiek prebehlo už v bode odoslania zloženého čísla smerom k službe na faktorizáciu čísel. Aktuálne v metóde získania výsledkov prevedieme zastavenie stopiek metódou Stop a následne vytvoríme objekt, ktorý reprezentuje tento presný časový interval. Nasledujúce kroky tvoria definíciu formátu výpisu tohto časového intervalu, kde v prvom prípade hovoríme o klasickom výpise času, obsahujúcom hodiny, minúty, sekundy a milisekundy. Druhý formát určuje celkový čas merania v sekundách, čo bolo využité pri získavaní hodnôt pre porovnanie aplikácií. Posledných pár krokov slúži k výpisu informácií do ListBox prvku, kde hlavnými sú: všetky faktory zvoleného čísla, definované zložené číslo, oba spôsoby zobrazenia času uvedené vyššie a počet klientov, ktorý sa na výpočte podieľali.

```
public void SendResults(BigInteger[] result)
{
    stopWatch.Stop();
    TimeSpan ts = stopWatch.Elapsed;
    string elapsedTime = String.Format("{0:00}:{1:00}:{2:00}.{3:00}",ts.Hours,
        ts.Minutes, ts.Seconds,ts.Milliseconds / 10);
    string elapsedTime2 = String.Format("{0:N10}", ts.TotalSeconds);
    Control.CheckForIllegalCrossThreadCalls = false;
    ResultList.Items.Clear();
    foreach (BigInteger d in result)
    {
```

```
            ResultList.Items.Add(d);
        }
        ResultList.Items.Add("---------------------------------");
        ResultList.Items.Add(multiplyList(result));
        ResultList.Items.Add("Time elapsed:" +elapsedTime);
        ResultList.Items.Add("Time elapsed:" + elapsedTime2);
        ResultList.Items.Add("Number of Slaves:" + SlaveList.Items.Count.ToString());
    }
```

6 VYHODNOTENIE VÝKONU NAVRHNUTEJ DISTRIBUOVANEJ APLIKÁCIE

Táto kapitola je založená na vzájomnej komparácii výsledkov aplikácie určenej k používaniu na jedinom zariadení a distribuovanej aplikácie pri zakomponovaní rôzneho počtu zariadení. V prvej časti si zhrnieme základné podmienky realizácie meraní. Následne si odôvodníme voľbu zložených čísel, na ktorých bude aplikácia testovaná a nakoniec sa budeme venovať zhodnoteniu nameraných výsledkov.

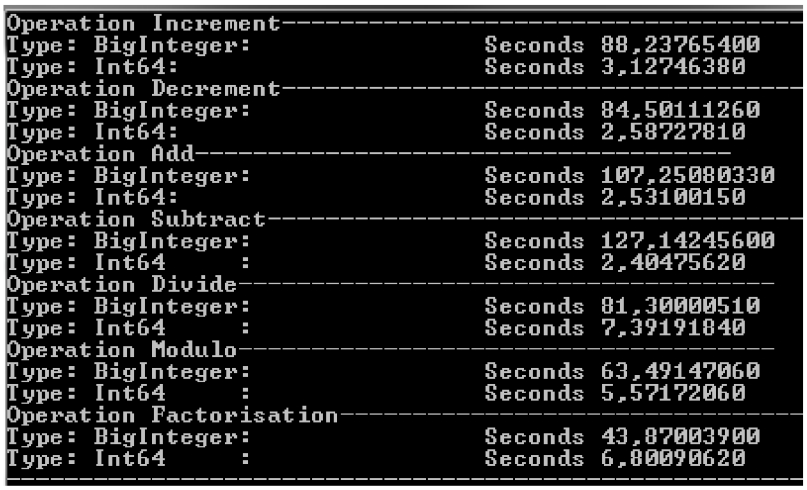
6.1 Podmienky a spôsob realizácie testov

Realizácia testov bola založená na opakovanom testovaní aplikácií na jednom až troch zariadeniach, pričom zariadenia neboli totožné. Prvým zariadením je notebook Toshiba P750-10Q s procesorom Intel Core i5-2410M CPU @ 2,30 GHz, ďalším je notebook LenovoIdeaPad S500 s procesorom Intel Core i7-3537U CPU @ 2 GHz a v poslednom prípade sa jednalo o notebook Asus K50IJ s použitým procesorom Intel Core 2 Duo CPU @ 2 GHz. Z toho dôvodu boli všetky merania realizované viacnásobne s rôznymi kombináciami prihlásenia zariadení a hodnoty získané v rámci meraní boli následne spriemerované. Zariadenia boli v prípade testovania distribuovanej aplikácie pripojené bezdrôtovo na rovnakú privátnu sieť.

Aplikácie využívajú k reprezentácii veľkých čísel triedu System.Numerics, ktorá implementuje typ BigInteger a jemu podliehajúce matematické operácie nad týmito nadmerne veľkými číslami. Využitie typu BigInteger však negatívne ovplyvňuje najmä efektivitu výpočtu faktorizácie čísel, a to je spôsobené zvýšenou zložitosťou operácií používaných pri realizácii algoritmu. V prípade, že tento dátový typ porovnáme s dátovým typom Int64, zisťujeme, že časová náročnosť je mnohonásobne väčšia. Oba dátové typy označujeme ako nemenné, čo sa prejavuje vytvorením nového objektu pri každej operácii prevedenej nad premennou daných typov. Typ BigInteger oproti Int64 disponuje výhodou v podobe možnosti využitia ľubovoľne veľkých čísel, kde neexistuje explicitne vyjadrená hranica. Pre možnosť porovnania dátových typov z hľadiska časovej náročnosti základných operácií bola vytvorená jednoduchá konzolová aplikácia. Obrázok 6.1 zobrazuje výsledný výpis aplikácie, na ktorom je možné porovnať konečné časy výpočtov. Finálne merania a následné časové porovnanie metód dokazuje, že použitie typu BigInteger výrazne ovplyvňuje časovú náročnosť ako základných operácií, tak konečnej faktorizácie čísel.

Samotné meranie výkonu aplikácií sa primárne zameriava na čas výpočtu faktorov zloženého čísla, kde počiatok merania predstavuje požiadavka v podobe definície nami zvoleného čísla a následného odoslania službe k realizácii výpočtu. Zakončením

meraného intervalu je potom bod, kedy riadiaci klient príjme výsledky faktorizácie v podobe zoznamu faktorov. K získaniu tohto intervalu môžeme s výhodou použiť triedu Stopwatch z menného priestoru System.Diagnostics, ktorá nám poskytuje vhodné metódy určené na presné meranie času počas behu aplikácie. Trieda následne umožňuje v rámci implementovaných metód rôzne konverzie a zobrazenia celkového času stráveného nad výpočtom. Túto skutočnosť sme využili k získaniu výsledných hodnôt celkového času výpočtu v sekundách.



Obr. 6.1: Porovnanie datových typov BigInteger a Int64.

Z hľadiska vierohodnosti výsledkov bolo nutné pristúpiť k voľbe optimálneho množstva meraní, ktoré aspoň čiastočne eliminujú mierne kolísanie výsledných hodnôt. Preto v prípade čísel zložených z dvoch faktorov s počtom cifier menším ako desať bol zvolený počet meraní sto a v prípade faktorov s počtom cifier prevyšujúcim desať prebehlo kvôli náročnosti výpočtu vždy desať meraní. Každý uvedený výsledok vychádza z aritmetického priemeru všetkých týchto meraní.

Pri meraní výkonu aplikácií sme sa zamerali na výpočty, ktoré sú známe svojou veľkou časovou náročnosťou. Obdobne ako pri algoritmoch založených na RSA sme testovali aplikácie na zloženom čísle, ktoré sa skladá z dvoch faktorov s definovaným počtom cifier, ktoré boli hodnotovo blízke, avšak nie rovnaké. Jednotlivé faktory boli zvolené náhodne zo stredu celého intervalu ohraničeného možnosťami definovanými počtom cifier. Spriemerované výstupy aplikácie boli následne vynesené do grafu uvedeného v ďalšej kapitole.

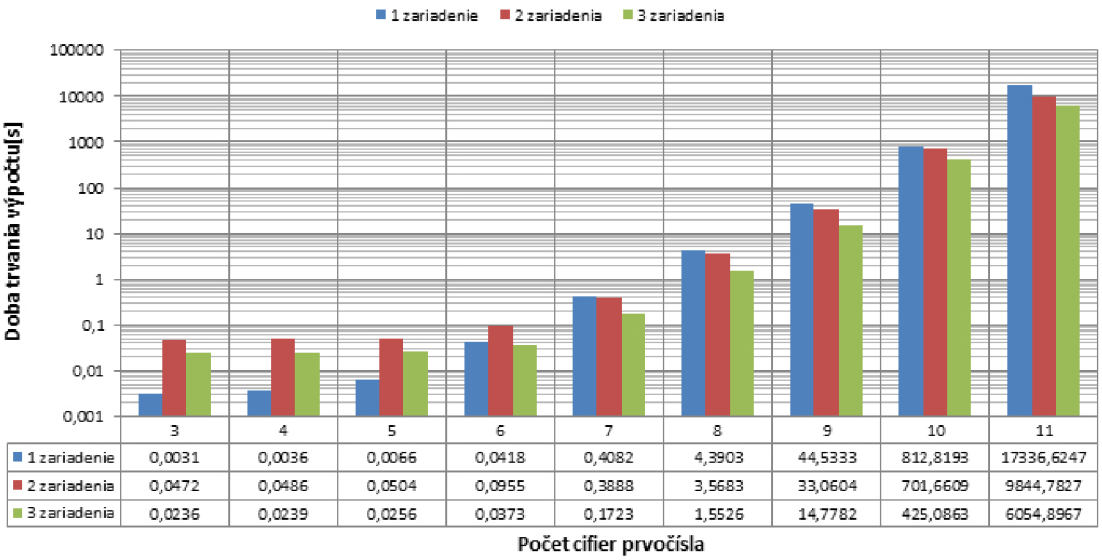
6.2 Zhodnotenie výsledkov meraní výkonu aplikácií

Na obrázku 6.2 vidíme výsledný graf (logaritmická mierka) nameraných hodnôt času v závislosti na voľbe čísla zloženého z faktorov s určeným počtom cifier. Z grafu môžeme odvodiť, že rýchlosť výpočtu faktorov s maximálnou dĺžkou do päť cifier je v prípade použitia ľubovoľného počtu zariadení približne rovnaká (v ráde desiatok až stoviek milisekúnd). V prípade aplikácie určenej jednému zariadeniu je vidieť, že pri voľbe čísel s menším počtom cifier (3 až 5) je výpočet dostatočne rýchly a môže tak z hľadiska výkonu konkurovať distribuovanej aplikácii. Táto skutočnosť je založená na zvýšenej réžii na pozadí distribuovanej aplikácie, kde je rozdelenie prehľadávaných intervalov medzi jednotlivé zariadenia, doplnkové operácie a ich komunikácia potenciálnym konzumentom výkonu. Preto sa v prípade voľby malých hodnôt (3 až 5 cifier) klady distribuovanej aplikácie v značnej miere neprejavujú. Z prvých štyroch meraní je jasné, že pri použití dvoch zariadení nie sme schopní kompenzovať výkon investovaný do réžie distribuovanej aplikácie, a tak je čas výpočtu osemnásobne (5 cifier) až pätnásťnásobne (3 cifry) väčší v porovnaní s jedným zariadením. V prípade varianty s troma zariadeniami pozorujeme zlepšenie, a teda je ich výkon už len štvornásobne (5 cifier) až osemnásobne (3 cifry) pomalší ako aplikácia pre jedno zariadenie. Zvýšenie veľkosti faktorov na šesť cifier predstavuje hranicu, kde aplikácia určená jednému zariadeniu svojim výkonom začína strácať výhodu eliminácie režijných výdajov. Z hľadiska výkonu pozorujeme vyrovnanie režijných nákladov pri použití troch zariadení s časom mierne rýchlejším v porovnaní s jedným zariadením. Zapojenie dvoch zariadení do výpočtu však naďalej nepostačuje na vyrovnanie réžie, a tak sa výpočet faktorov pohybuje okolo dvojnásobku času jedného zariadenia.

Použitie hodnoty faktorov so siedmimi ciframi je zlomovým bodom, v ktorom začína prevažovať výhoda voľby distribuovanej aplikácie. Výpočet pomocou dvoch zariadení prináša úsporu času v podobe 5 % v porovnaní s časom výpočtu jedného zariadenia. Tri výpočtové prvky svojim výkonom dosiahli až 57 % úspory času, ktorá predstavuje značný rozdiel s aplikáciou pre jeden prvok. Neustále sa však jedná o výpočty spadajúce pod polovicu sekundy, ktoré užívateľ pri používaní aplikácie výraznejšie nepociťuje.

Pri pokračujúcom náraste faktorov na sedem až jedenásť ciferné je možné pozorovať subexponenciálny rast zložitosti výpočtu. Faktorizácia sa stáva značne časovo náročnou najmä pri použití jediného zariadenia. Rozdiely medzi aplikáciami sa už výraznejšie nemenia a výhoda použitia viacerých zariadení spočíva v úspore času pri použití dvoch zariadení v rozsahu 15 až 45 %. Pri zapojení troch zariadení do výpočtu urýchlime výpočet faktorizácie o 45 až 70 %.

Výsledky získané počas testovania aplikácií dokazujú skutočnosť, že distribuovaná aplikácia úspešne rozdeľuje záťaž na všetky pripojené zariadenia. Tým sme schopní redukovať časové náklady na výpočet faktorov čísel. Na základe týchto predpokladov môžeme tvrdiť, že pri zainteresovaní ešte väčšieho množstva zariadení ako pri testovaní dokážeme výpočet zjednodušiť natoľko, že by sme boli schopní faktorizovať zložené čísla rádovo vyšších faktorov v reálnom čase.



	3	4	5	6	7	8	9	10	11
1 zariadenie	0,0031	0,0036	0,0066	0,0418	0,4082	4,3903	44,5333	812,8193	17336,6247
2 zariadenia	0,0472	0,0486	0,0504	0,0955	0,3888	3,5683	33,0604	701,6609	9844,7827
3 zariadenia	0,0236	0,0239	0,0256	0,0373	0,1723	1,5526	14,7782	425,0863	6054,8967

Obr. 6.2: Výsledky meraní výkonu aplikácií.

7 ZÁVER

Úlohou mojej diplomovej práce bolo získanie nadhľadu a spísanie teoretických základov venujúcich sa distribuovaným systémom a technológii WCF, určenej na návrh distribuovaných aplikácií. Ďalšou podstatnou časťou diplomovej práce bolo využitie informácií o WCF k vytvoreniu príkladov, ktoré demonštrujú možnosti tejto technológie. Jadro celej práce však tvoril samotný návrh distribuovanej aplikácie, ktorej hlavnou úlohou je riešenie problému faktorizácie čísel.

Prvá časť diplomovej práce má za úlohu vysvetliť základy distribuovaných systémov, ich charakteristické vlastnosti, klady a zápory, problémy, s ktorými sa pri distribuovaných systémoch stretávame, a taktiež modely distribuovaných systémov.

Ďalšou podstatnou zložkou práce sú kapitoly venované frameworku WCF a následne praktickým ukážkam schopností tejto technológie. Druhá kapitola tak obsahuje možnosti, ktoré WCF ponúka a ich výhody pri použití tejto technológie na návrh distribuovaných aplikácií. V ďalšej časti sa zameriavame na popis princípu service-oriented architecture, na ktorom je WCF postavené. Posledná najdôležitejšia časť kapitoly slúži ako vstup do tvorby WCF aplikácií, kde sa venujeme najskôr podstatným prvkom vývoja samotnej WCF služby, následne riešeniu hosťovania vytvorenej služby a v závere konštrukcii klienta, ktorý bude službu využívať.

Tretia kapitola popisuje klasické metódy faktorizácie čísel, kde faktorizáciou označujeme proces rozkladu zloženého čísla na konečný súčin prvočísel. Tieto metódy sú následne podrobne popísané a v zjednodušenej forme implementované v prostredí C#.

Štvrtá kapitola je venovaná praktickej implementácii informácií z teoretického úvodu, a to v podobe jednoduchých príkladov, ktoré vysvetľujú všetky podstatné časti vytvorenia WCF aplikácií.

Prvý príklad sa zaoberá základnými stavebnými kameňmi WCF služby, ako sú napr. ServiceContract, DataContract atď. Podstatnou zložkou prvej ukážky je podrobný popis vytvorenia hosťovania našej služby. Posledným dôležitým prvkom pri vytvorení funkčnej WCF aplikácie je klient, ktorý bude službu využívať vo svoj prospech. Aj o ňom sa diskutuje v prvej ukážke.

Druhý príklad je venovaný komunikačným vzorom, ktoré WCF poskytuje, opäť s praktickou ukážkou a komentármi, ktoré vysvetľujú spôsob chovania jednosmerného, otázka/odpoveď a duplexného vzoru.

V treťom príklade sú vysvetlené inštančné módy WCF. Tie charakterizujú spôsob chovania WCF služby pri volaní klienta. Príklad venovaný tejto problematike zobrazuje chovanie všetkých troch inštančných módov, a tými sú PerCall, PerSession a Single mód.

Hlavným výstupom diplomovej práce je však distribuovaná aplikácia určená na

riešenie faktorizácie čísel, ktorej základnou úlohou je dosiahnuť úsporu času pri riešení tohto zložitého problému. Návrh aplikácií je bližšie špecifikovaný v piatej kapitole, kde v prvej časti je rozobratá problematika voľby vhodného faktorizačného algoritmu pre naše účely. Táto časť tiež obsahuje modifikácie, ktoré sme použili na pôvodný algoritmus uvedený v teoretickom úvode. V ďalšom bode práce sa oboznámime s hlavnými prvkami celej architektúry distribuovanej aplikácie, ktorými sú WCF služba, výpočtový klient a riadiaci klient. Dozvedáme sa o primárnych úlohách každého z týchto prvkov. Nemenej dôležitým elementom fungovania distribuovanej aplikácie je komunikácia medzi jednotlivými účastníkmi, ktorou sa zaoberá ďalšia časť práce. V tejto časti zisťujeme všetky podstatné informácie, týkajúce sa komunikácie v rámci distribuovanej aplikácie. Prevažne sa zameriavame na odôvodnenie voľby správneho bindingu pre naše potreby, ktorý definuje spôsoby komunikácie a kódovania vymieňaných správ. Ďalej sú predstavené dôležité aspekty uvedenia distribuovanej aplikácie do chodu pri reálnom nasadení na viacerých zariadeniach. Pri nasadení aplikácie do reálnej prevádzky je nutné oboznámiť užívateľa, ktorý bude aplikáciu využívať, so základnými komponentmi GUI, s ktorými môže prísť do kontaktu. Tejto úlohy sa zhosťuje ďalšia časť práce, v ktorej postupne rozoberáme chovanie grafických prvkov aplikácie. Od významu základných výpisov každej z aplikácií po podrobný rozbor funkcie daného prvku. Po prečítaní tejto časti by tak mal mať užívateľ dostatok informácií na to, aby distribuovanú aplikáciu dokázal ovládať a interpretovať jej výstupy. Zostávajúce dve časti práce predstavujú prienik do spôsobu realizácie celej aplikácie, kde sa najskôr stretáme so zjednodušeným sekvenčným diagramom fungovania hlavného poslania aplikácie, ktorým je faktorizácia zloženého čísla. Môžeme sa tak oboznámiť so základnými metódami, slúžiacimi k riešeniu zadaného problému, a tiež s vecnými radami, ktoré je nutné použiť na prvky aplikácie, aby reagovali, napr. na mnohonásobné volania entít v určitom momente. Analýza funkcionality distribuovanej aplikácie je zavŕšená komentovanými ukážkami kódu, ktoré vysvetľujú postupy a definície vybraných metód. Jedná sa najmä o úseky kódu, ktoré majú vysokú prioritu pri riešení problému faktorizácie na strane služby, výpočtového a riadiaceho klienta.

Posledná kapitola diplomovej práce je dôležitá z hľadiska porovnania vytvorených aplikácií, kde prvá bola určená výhradne jednému zariadeniu a druhá, distribuovaná aplikácia, bola testovaná pri zapojení dvoch a troch zariadení kooperujúcich na výpočte zhodného algoritmu faktorizácie. Výsledky merania výkonu aplikácií nám odhaľujú, že v prípade použitia aplikácie určenej jednému zariadeniu sme dosiahli výborné výsledky pre prípady čísel, ktoré pozostávali z faktorov do šiestich cifier. To je spôsobené zvýšenou réžiou, ktorú vyžaduje k svojmu behu distribuovaná aplikácia. Naopak pri väčších hodnotách zloženého čísla bola potvrdená výhoda použitia distribuovanej aplikácie, čím bola dosiahnutá značná úspora času v porovnaní s jed-

ným zariadením. Táto úspora dokazuje, že vytvorená aplikácia úspešne distribuuje problém medzi viacero zariadení, čo má za následok zvýšenie výkonu pri riešení faktorizácie. Výsledky meraní tiež preukazujú, že nárast výkonu je priamo závislý na počte zariadení prihlásených ako výpočtový klient.

Na záver môžeme konštatovať, že technológia WCF prináša výrazné možnosti a výhody z hľadiska návrhu a implementácie distribuovaných aplikácií. Tieto aplikácie dokážu prispieť k zrýchleniu komplexných výpočtov, a tým prinášajú značné možnosti pri riešení úloh, ktoré by v prípade použitia jedného zariadenia nebolo možné uskutočniť v reálnom čase.

LITERATÚRA

[1] SCHLICHTER, J. *Distributed Applications.* Germany: Technische Universität München, 2002.

[2] *MODELS OF DISTRIBUTED SYSTEMS.* [online]. Department of Computer and Information Science LiU, 2007. Dostupné z URL: <`https://www.ida.liu.se/~TDDB37/lecture-notes/lect2-3.frm.pdf`>.

[3] DRAGONI, N. *Distributed Systems: Models and Design.* [online]. Department of Informatics and Mathematical Modelling DTU, 2013. Dostupné z URL: <`http://www2.imm.dtu.dk/courses/02220/2013/L1/L1_02.pdf`>.

[4] *System Models.* [online]. Department of Computer Science University of Crete, 2012. Dostupné z URL: <`http://www.csd.uoc.gr/~hy556/material.html`>.

[5] KLEIN, S. *Professional WCF Programming .NET Development with the Windows® Communication Foundation.* USA: Wiley Publishing, Inc., 2007. ISBN 978-0-470-08984-2.

[6] CIBRARO, P.; CLAEYS, K.; COZZOLINO, F.; GRABNER J. *Professional WCF 4: Windows Communication Foundation with .NET 4.* USA: Wiley Publishing, Inc., 2010. ISBN 978-0470563144.

[7] LÖWY, J. *Programming WCF Services.* 3. vyd. USA: O'Reilly, Inc., 2007. ISBN 978-0596526993.

[8] CONNELLY, B. *Integer Factorization Algorithms.* [online]. USA: Oregon State University, 2004. Dostupné z URL: <`http://connellybarnes.com/documents/factoring.pdf`>.

[9] CRANDALL, R.; POMERANCE, C. *Prime Numbers.* 2. vyd. USA: Springer, 2005. ISBN 978-0-387-25282-7.

ZOZNAM SYMBOLOV, VELIČÍN A SKRATIEK

ASMX active server methods

ASP active server pages

FTP file transfer protocol

HTTP hypertext transfer protocol

IIS internet information services

IPC inter process communication

MSMQ message queuing

P2P peer to peer

PHP hypertext preprocessor

RPC remote procedure call

SOA service-oriented architecture

SOAP simple object access protocol

TCP transmission control protocol

UDP user datagram protocol

URL uniform resource locator

WAS windows activation service

WCF windows communication foundation

WPF windows presentation foundation

WSDL web service description language

WSE web service enhancements

XML extensible markup language

ZOZNAM PRÍLOH

A OBSAH PRILOŽENÉHO DVD

- Elektronická verzia diplomovej práce v PDF (xkisac01.pdf)
- Projekty distribuovanej aplikácie (BigIntPrimeService, BigIntMasterClient, BigIntSlaveClient)
- Konzolová aplikácia na faktorizáciu čísel pre jedno zariadenie (PrimeNumber)