Distribuovaný evoluční algoritmus na platformě .NET

Distributed evolutionary algorithm using .NET platform

Bc. Petr Rympler

Diplomová práce
2010

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

*** nascannované zadání str. 1 ***

*** nascannované zadání str. 2 ***

ABSTRAKT

Cílem této diplomové práce je vytvoření jednoduchého, avšak v budoucnu komplexně rozšířitelného software na platformě Microsoft .NET, který bude umožňovat distribuovat jednotlivé úlohy a zpracovávat je. V teoretické části budou probrány důležité témata pro porozumění možnostem platformy .NET a možnosti distribuovaných aplikací. V praktické části se zaměříme na samotný programu a distribuci zadání.

Klíčová slova: Distribuované evoluční algoritmy, platforma Microsoft .NET, WCF

ABSTRACT

The main focus of this project is to create simple but complexly extensible software on the Microsoft .NET platform which will allows distributing simple tasks and execute them. In first part we will focus on .NET platform features and distributed computing features. In second part we will mainly focus on core program and task distribution.

Keywords: Distributed evolutionary algorithm, Microsoft .NET platform, WCF

Chtěl bych poděkovat svojí rodině, přítelkyni a svým kamarádům za dlouholetou podporu, bez které se nemůže nikdo obejít. Taktéž chci poděkovat svému vedoucímu diplomové práce Ing. Bc. Pavlovi Vařachovi.

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně …….

podpis diplomanta

OBSAH

ÚVOD

Dávno pryč jsou již doby, kdy složité výpočetní operace, kalkulace a řešení matematických problémů lidstvo nemělo jak řešit. Řada matematických i fyzikálních jevů tak čekala až budoucnost, kdy budou k dispozici prostředky na řešení více či méně komplexních úloh. S rozvojem počítačů v 20. století, kdy děrné štítky nahradili modernější, kapacitnější média a z nynějšího pohledu jednoduchých sálových počítačů o rozměrech fotbalového hřiště se staly osobní počítače, již není řešení dříve požadovaných výpočetních operací nereálné.

Distribuované výpočty mají své kořeny již v počátku 21. století, kdy zcela běžný a nijak zkušení uživatelé měli a stále mají možnost zapojit se do řešení různých problémů v podobě hledání léku rakoviny, AIDS nebo zcela odlišné hledání mimozemské civilizace analýzou datových vzorků.

Fenoménem dnešní doby se stávají distribuované výpočty a operace, které pomáhají řešit komplexní úlohy, jenž by za pouhého použití superpočítačů trvalo řadu až stovky let. Díky možnostem internetu a počtu počítačů do něho připojených je analýza vzorků mnohem rychlejší. Samotný desktop počítač patřící běžnému uživateli nemůže nijak konkurovat velkým superpočítačům po celém světě. Pokud však vezmeme v úvahu již zmíněné možnosti internetu a množství počítačů do něho připojených, získáme v globálním měřítku ohromný výpočetní výkon, se kterým se dá již ledacos podniknout.

Budoucnost internetu patří distribuovaným aplikacím, projektům a rozsáhlým sítím. Dalším rýsujícím fenoménem je Cloud computing, na který se v této práci také zaměříme.

I. TEORETICKÁ ČÁST

1 DISTRIBUOVANÝ SYSTÉM

Je takový systém, který se skládá (je propojen) z množiny nezávislých uzlů, který navenek udává dojem jednotného a uceleného systému.

Hlavním cílem distribuovaného systému je připojení uzlů transparentním, otevřeným a stupňovaným způsobem. Takto navržený systém dokáže odolávat vůči výpadkům a je obecně výkonnější.

1.1 Typy distribuovaných systémů

- **Klient – server** – jedná se o typ komunikace, kdy klient kontaktuje server, který provádí danou distribuovanou úlohu.
- **Třístupňová architektura** – klientská aplikace je propojená se serverovou aplikací, která je následně spojená s databází.
- **N-stupňová architektura** – jedná se o třístupňovou architekturu rozšířenou o další vrstvy.
- **Clusterová architektura** – množina strojů pracujících paralelně na nějakém procesu (úloze). Úloha je rozdělena na části, kdy každá část je vykonána na jednom stroji a na závěr je spojena do jednoho společného výsledku.
- **Peer-to-peer** – architektura, kde neexistuje žádný řídící prvek, který spravuje zdroje. Vše je rozděleno mezi klienty (stroji) bez nutnosti využití serveru.
- **Space-based** - síťová infrastruktura vytvářející iluzi jednotného adresového prostoru. Data jsou transparentně replikována podle potřeb aplikace.

1.2 Nevýhody distribuovaných systémů

Mezi hlavní nevýhody distribuovaných systémů patří:

- Každá úloha, která je součástí distribuovaného výpočtu, musí být předem připravena (přizpůsobena) na použití v distribuovaných systémech. Špatnou analýzou nebo realizací úlohy dochází k nesprávné či zcela špatné interpretaci výpočtu, který se stává nepoužitelný.

- V případě špatně navržené úlohy v rámci distribuovaného systému je složitější diagnostika a řešení problému. Případná porucha může vyžadovat nutnost připojit se na cílový počítač (uzel) a zkoumat problém v daných konkrétních podmínkách.
- Špatně navržený systém může vést k nestabilitě, pádu jednotlivých uzlů a problémům, se kterými se setkáme realizací systémů.
- Nesprávná volba komunikačního kanálu, jeho kódování (zabezpečení) a odolnost vůči poruchám může výrazným způsobem ovlivnit dostupnost celého systému.

1.3 Distribuovaný výpočet

Distribuovaný výpočet je výpočet rozdělený na více menších, méně náročných úloh za účelem rychlejšího vyřízení požadavku předaného programu. Lze ho využít jen u výpočtů, jejichž algoritmus lze paralelizovat – převést na paralelní verzi, kdy vzájemně nezávislé části výpočtu běží současně.

Výpočet lze distribuovat buď na úrovni operačního systému s přesměrováním softwarových vláken na jiné členy clusteru, nebo přímo v režii programu, který se nainstaluje v podobě mnoha klientů na každý z počítačů tvořících cluster.

Mezi typické výpočty, které je vhodné řešit distribuovaně, patří analýza velkého množství statistických dat, „zpětné inženýrství“ DNA, modelování struktury proteinů, generování fraktálů nebo zkoumání vesmírného vlnění na přítomnost rádiového signálu mimozemšťanů (SETI). [1]

1.4 Distribuované projekty

V současné době existují stovky a možná i tisíce nejen vědeckých projektů, které prostřednictvím daného distribuovaného systému provádí předem definované výpočetní operace. Díky existenci internetu a množství osobních počítačů do něho připojených roste také výpočetní výkon, který může být využit na řešení daného distribuovaného projektu. Pokud vezmeme v úvahu počet v současné době připojených počítačů a výkon každého jednotlivého počítače, který není ani zdaleka využit, získáme výpočetní výkon v řádu tisíců TFLOPS, který předčí i ty největší superpočítače na světě. Výkon současného nejrychlejšího superpočítače Jaguar Cray XT je v tuto chvíli 1759 TFLOPS a výkon sítě BOINC je v tuto chvíli přes 5000 TFLOPS. [2]

1.4.1 Projekt SETI@home

Za otce distribuovaných projektů je považován projekt SETI@home hledající mimozemskou civilizaci, který vznikl v 50. letech minulého století. Tento projekt postupně zahrnoval spoustu projektů pro vyhledávání mimozemských civilizací ve vesmíru dostupnou technikou. Po odchodu NASA v roce 1993 ztratil projekt obrovské finanční prostředky a musel se vydat cestou úspor. Předcházející rádiové SETI výzkumy používaly speciální, za tím účelem navržené, superpočítače umístěné přímo v teleskopu které prováděly převážnou část celé analýzy dat. Jejich výkon byl sice obrovský, ale nedostačující. Právě zde se zrodila myšlenka distribuovaných výpočtů.

David Gedye s Craigem Kasnoffem v roce 1995 navrhli, aby se radiový výzkum SETI prováděl na virtuálním superpočítači, který by byl složen z velkého množství individuálních počítačů připojených na Internet, a zorganizovali tak projekt SETI@Home, který si dal za cíl tento nápad prozkoumat. Tito dva počítačoví vědci ze Seattlu přišli s geniálním řešením pomocí již zmiňovaných distribuovaných výpočtů. Namísto jednoho drahého superpočítače, který by běžel dlouhou dobu, chtěli využít tisíce běžných počítačů, které by pracovaly jen krátký časový úsek. Jejich představou bylo využít obrovské popularity projektů SETI mezi širokou veřejností. Lidé po celém světě by pak měli možnost stáhnout si prostřednictvím sítě Internet analyzující program na své osobní počítače, přičemž program by po instalaci a následném spuštění nikterak nezasahoval do běžné práce a běžel by pouze po určitý čas. Jakmile by přijatá data byla zpracována, program by je odeslal zpátky na centrální server a stáhl by si další data ke zpracování. Trvalo ovšem téměř 5 let než se za pomoci Planetární společnosti podařilo tuto geniální myšlenku uvést do praxe.

V roce 1996 sestavili David Gedye s Craigem Kasnoffem svůj vlastní počítačový projektový tým a představili vědecký plán, který byl přijat akademickou obcí na 5. mezinárodní konferenci bioastronomie v červenci 1996 v italském Capri. V následujícím roce 1997 projektový tým dokončil prototypy klientského a serverového software a kódu analyzujícího přijatá data. Rok 1998 byl prakticky celý věnován otestování a důkladnému ověření funkčnosti celého systému. Na jaře 1998 se Planetární Společnost stala hlavním finančním sponzorem projektu SETI@Home, který již byl pod vedením Davida Andersona a Dana Werthimera.

Začátkem září 1998 začal pracovat systém přijímající, ukládající a distribuující data a byla také dokončena finální verze klientského software, která již mohla být nasazena k veřejnému použití. Počátkem roku 1999, zhruba do dubna, byla otestována a odladěna finální verze klientského software a ke spuštění byly připraveny i internetové stránky projektu. Ke spuštění projektu SETI@Home došlo přesně 17. května 1999 a tento den se zapsal do historie SETI zlatým písmem. Během pár měsíců se do projektu zapojily statisíce lidí po celém světě a již v srpnu roku 1999, tedy po pouhých 3 měsících fungování, projekt SETI@Home měl přes 1 milión uživatelů. Počátky byly sice trošku krkolomné, protože datové servery nestíhaly reagovat na takové množství požadavků, ale postupem času a za velkého úsilí se podařilo většinu technických problémů odstranit.

Obr. 1 Satelit sloužící k příjmu vzorků z vesmíru [3]

Po roce fungování projekt SETI@Home pokořil další metu, kterou byl celkový počet dva milióny uživatelů, z čehož aktivních jich bylo přibližně půl miliónu. Mezi aktivní uživatele se počítali účastníci, kteří za poslední měsíc odeslali alespoň 1 zpracovaný datový balíček. Začal náročný úkol třídění více než 1.4 biliónu potenciálních signálů z databáze, eliminace

veškerého rádiového rušení, vyhledávání chyb při zpracování a nalezení opakujících se signálů. Projekt se dále rozšiřoval na základě vzrůstajícího zájmu po celém světě. Do původního projektu SETI@Home, označovaného nyní jako Classic, se zapojilo více než 5 milionů uživatelů ze všech možných koutů celého světa. [4]

1.4.2 Projekt BOINC

Projekt BOINC nabízí infrastrukturu pro běh projektů jako SETI@home, kdy umožňuje badatelům z různých oborů jako astrofyzika, klimatologie nabídnout dostatečně velký výpočetní výkon prostřednictvím připojení do sítě připojeních PC. Projekt BOINC je vyvíjen skupinou lidí z Kalifornské univerzity v Berkeley, která je vedena již zmíněným Davidem Andersonem. Celý projekt je uvolněn jako open-source pod licencí LGPL.

Díky tomuto projektu se vysokým výpočetním výkonem zpracovávají úlohy v oblastech hledání struktury bílkovin, proteinů, rozklad čísel na prvočísla nebo také projekty pro renderování náročné počítačové grafiky.

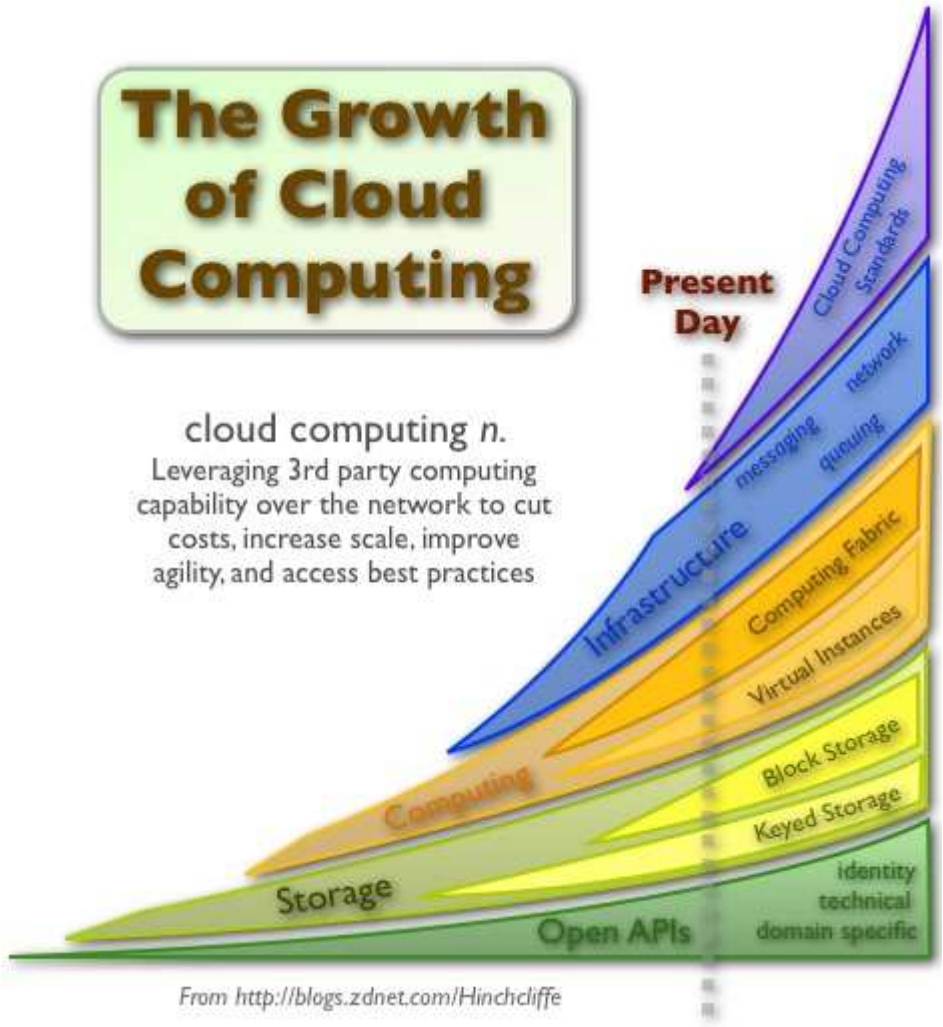
Součástí celé sítě BOINC jsou jednotlivě zapojené počítače pomocí staženého klienta a serverů, které pouze distribuují výpočetní jednotky.

2 CLOUD COMPUTING

Je na internetu založený model vývoje a používání počítačových technologií, kdy hardwarové i softwarové prostředky jsou sdíleny v síti internet. Cloud computing popisuje nový model doručování IT služeb koncovému zákazníkovi i společnostem. Typičtí poskytovatelé cloud computingových služeb nabízejí různé business aplikace online, kdy zákazníci k nim přistupují pomocí internetu ze svého prohlížeče, vlastní aplikace. Všechny tyto aplikace jsou hostovány na rozsáhlé síťové infrastruktuře hostované v různých datacentrech.

Zákazník jako takový nevlastní žádnou infrastrukturu a vše si kompletně pronajímá přímo od poskytovatele. Díky povaze služeb zákazník platí pouze za využité prostředky bez nutnosti finančních investic a utrácení do prostředků, které nevyužijí.

Výhodou celé této infrastruktury je při srovnání s klasickým modelem možnost využití větších prostředků, byť jen špičkově, což by u klasické služby nebylo možné. Pro jednodušší představu si můžeme představit provozování internetového obchodu. Pokud provozujeme daný internetový obchod na sdíleném hardware, můžeme narazit na jeho limit v podobě špičky, kdy především v době vánočních svátků dochází ke zvýšení návštěvnosti v řádu stovek procent. Takto provozovaný internetový obchod s vysokou návštěvností by tedy na sdíleném hardware nestíhal a naopak v průběhu roku byl nadbytečně naddimenzovaný. Díky řešení cloud computingu můžeme využít možnost lineárního růstu zátěže díky charakteru služby, která může být kdykoliv navýšena. Díky tomuto modelu platíme skutečně pouze za to, co využijeme.



Obr. 2 Popis růstu Cloud computingu a jeho složení [5]

2.1 Srovnání

Cloud computing vychází z několika následujících typů, avšak nemůže být s nimi zaměňován:

- **Autonomic computing** – iniciativa vytvořená IBM v roce 2001, kdy cílem je vyvinout počítačové systémy schopné udržování sama sebe. Díky nynějšímu nezadržitelnému růstu počítačových technologií a systémů dochází ke zvyšování nároků na správu a údržbu. Takto vytvořený systém rozhoduje sám za sebe (provádí vlastní rozhodnutí) a automaticky se adaptuje nově vzniklým událostem a podmínkám.

- **Klient-server model** – tento model všeobecně popisuje distribuované aplikace mezi klientem a serverem.
- **Grid computing** – jedná se o formu distribuovaných výpočtů a paralelních výpočtů, kdy množina serverů (strojů) je spojena do jednoho velkého virtuálního počítače, který vykonává zpracování výpočtu.
- **Mainframe** – vysoce výkonné počítače využívané nadnárodními společnostmi pro řešení náročných úkonů. Typickým příkladem může být zajištění běhu finančních transakcí, generování statistik a další plánování zdrojů.
- **Utility computing** – počítá využití zdrojů jako procesorový čas, diskový prostor podobně jako například odběr elektřiny. Tento systém počítání využití zdrojů vyniká tím, že nemusíme investovat do počáteční infrastruktury.
- **Peer-to-peer** - již zmíněná architektura, kdy není vyžadován centrální prvek. [6]

2.2 Typy cloud computingu

- Vyhrazený cloud (Dedicated cloud) - jedná se o vyhrazené kapacity datových center, kde organizace provozují své klíčové aplikace, kde jakýkoli únik dat by byl pro firmu naprosto kritický a zničující.
- Privátní cloud (Private Cloud) - tento typ cloudu je podobný jako ten předchozí, ale využívá se především pro vývoj a testování. Rozdíl mezi vyhrazeným cloudem je v tom, že více aplikací sdílí stejné výpočetní zdroje.
- Sdílené prostředí (Shared Environment) - sdílené prostředí je výpočetní výkon k dispozici pro určité typy aplikací, kde jsou předem známé nároky na výpočetní výkon, tedy je známo, kdy bude která aplikace spotřebovávat kapacity serverů. Servery mohou být určené pro jednu firmu nebo pro více, které používají stejnou aplikaci apod.
- Veřejný cloud (Public Cloud) - toto jsou služby, které si představuji pod klasickým cloudem, jsou to služby Amazonu, Salesforce.com a dalších, tedy sdílený výpočetní výkon je pronajímán komukoli, kdo projeví zájem. Zatímco na všech třech předchozích typech se provozují standardní klient - server aplikace, které známe z

podniků dnes, tak poslední typ přináší SaaS aplikace a skutečnou změnu technologického paradigmatu. [7]

2.3 CaaS, SaaS a ostatní

Cloud computing lze zjednodušeně rozdělit na dvě hlavní součásti, CaaS (Communication as a Service) a SaaS (Software as a Service). Tyto dvě nelogicky znějící zkratky v podstatě představují základní dělení cloud computingu a ke svému provozu využívají další služby, konkrétně IaaS (Infrastructure as a Service), MaaS (Monitoring as a Service) a PaaS (Platform as a Service). Spousta zvláštně znějících názvů, ve skutečnosti ale skrývá poměrně jednoduché a snadno pochopitelné principy.

Jak už bylo zmíněno, na vrcholku pomyslné pyramidy stojí služby CaaS a SaaS. V případě prvního jmenovaného se jedná o využití technologie cloud computingu pro zajištění komunikačních potřeb zákazníka. Pod tím se může skrývat cokoliv od zajištění mailserverů, přes VoIP komunikaci, až po provoz firemních kolaboračních nástrojů. Služba je přitom stejně jako hardware a jeho provoz zajištěna poskytovatelem služby a zákazník ji tak pouze využívá a nestará se o ni.

Druhým zmíněným pojmem je Saas, Software as a Service, neboli software jako služba. V rámci ní si zákazník v podstatě pronajímá vybraný software od poskytovatele, přičemž ten je mu dodán přesně na základě jeho požadavků, nikoliv tedy rozsáhlejší řešení, které by pro jeho nároky bylo příliš velké. V rámci SaaS lze ovšem získat přístup k aplikacím prakticky libovolného rozsahu, od základního textového editoru a emailového klienta až po rozsáhlá CRM řešení. Především v případě rozsáhlejších řešení platí, že jejich získání formou SaaS bývá v dlouhodobém horizontu levnější, než pořízení a nasazení vlastní verze.

Výhodou SaaS oproti klasickému, krabicovému, softwaru jsou také jeho častější aktualizace. Ty probíhají prakticky okamžitě a v jednom sledu je aktualizován software používaný všemi uživateli. Přístup k aplikacím obvykle probíhá prostřednictvím internetového prohlížeče, což eliminuje nutnost jakýchkoliv instalací a přidává tak další výhodu cloud computingu jako takového, a to jednoduchost. Platby za využívání SaaS mohou být v zásadě trojího typu, a to podle počtu uživatelů, podle rozsahu požadavků (např. počet databází, apod.) a podle míry používání.

Jak už bylo zmíněno dříve, jak Caas, tak i Saas využívají ke svému provozu některé další služby. Konkrétně se jedná především o PaaS, mohou sem ale spadat i zmíněné služby IaaS a MaaS. Nemá cenu zabíhat do detailů, pojďme jen shrnout hlavní rozdíly mezi těmito třemi přístupy. V rámci služby IaaS si má zákazník možnost pronajmout virtuální počítač, nebo server. Virtuální proto, že pronajímány nejsou většinou de facto jednotlivé počítače, ale pouze části výkonu nějakého většího celku. Zákazník si poté může zvolit, jak bude s daným výkonem nakládat, je také ovšem na něm, aby si vlastními silami nainstaloval veškerý software. V případě IaaS se výška platby vypočítává buď podle velikosti požadovaného výkonu, případně času, po které byl výkon využíván, často ovšem také pomocí koeficientu, který porovnává obě tyto hodnoty.

Na rozdíl od IaaS je v rámci služeb PaaS alespoň v krátkodobém horizontu nutné počítat s omezením v podobě předvolené platformy. Na tu lze nahrát, zprovoznit a používat libovolné přičemž jejich zajištění a implementace je opět na zákazníkovi samotném. Princip práce se službou je pak velmi podobný klasickému internetovému hostingu. Podobně jako v případě IaaS jsou i u PaaS platby založeny na využitém výkonu platformy. Špatnou zprávou je ovšem poměrně malá dostupnost těchto služeb, poskytuje je například Google v rámci svého Google App Engine.

O trh cloud computingu je ze strany poskytovatelů velký zájem. Posledním zástupcem jsou služby MaaS, neboli Monitoring as a Service. Jak už název napovídá, hlavním smyslem těchto služeb je monitoring a vyhodnocování dat ve všech představitelných podobách. Služby MaaS jsou poměrně důležité pro všechny ostatní a dříve zmíněné typy služeb, protože poskytují komplexní přehled o fungování jednotlivých aplikací, jejich výpočetních potřebách, stejně jako o využití výkonu, který je k dispozici. [8]

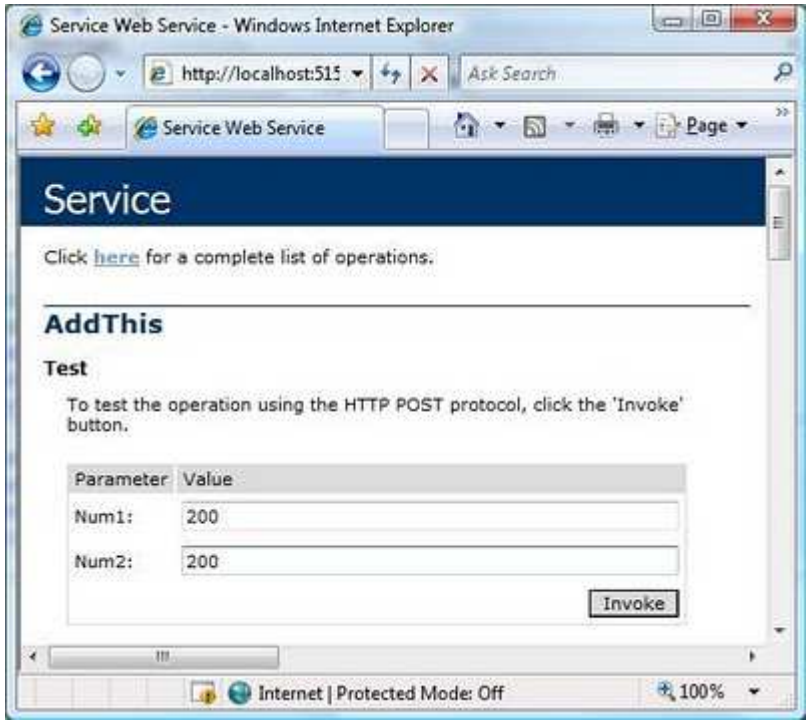
3 MOŽNOSTI KOMUNIKACE A DISTRIBUCE NA PLATFORMĚ MICROSOFT .NET

3.1 ASP.NET Web Services

Webová služba je podstatou internetu, tak jak ho známe a používáme. Přestože sám pojem webová služba (Web Service) se objevilo teprve nedávno, jeho podstata není žádnou novinkou. Webovou službu si můžeme přestavit jako abstraktní rozhraní, poprvé uplatněné při HTML interakci mezi uživatelem a webovým serverem. Když uživatel zadá nějakou adresu URL, volá tak výchozí metodu modulu na straně serveru, který poskytuje určité služby. Návratovou hodnotou je stránka ve formátu znakového HTML, kde jsou smíšeny textové informace s grafikou a údaji o rozvržení stránky.

Zkusme tuto koncepci rozvinout trochu hlouběji a získáme tak obecně uznávanou definici webové služby. Webová služba je softwarová aplikace, dostupná jiným aplikacím přes web. Webovou službu můžeme nasadit v libovolném typu webového prostředí: internetu, intranetu nebo extranetu. Pro její nalezení a přístup k ní stačí uživateli jediné – její URL adresa. Teoreticky může k této adrese přistupovat více internetových protokolů. V praxi je však protokolem pro přístup k webovým službám vždy HTTP. Webová služby má pouze jeden druh zákazníka: jinou aplikaci. Uživatelé samozřejmě ke službě přistupovat mohou, ale vždy jen pomocí prostředků nějaké softwarové aplikace.

Nedávné soustředění softwarové komunity na sám pojem „webová služba“ naznačuje, že technologie – nezávisle na použité platformě nebo filozofickém přístupu – je již dostatečně propracovaná na to, aby bylo možné zobecnit a parametrizovat softwarový mechanismus přístupu k adrese URL. Webové služby mají rozšířené a mnohem výkonnější programovací rozhraní pro přístup k webovým aplikacím. Namísto jediné výchozí metody pro přístup k URL a zpětnému zaslání prostého kódu HTML můžete nyní použít spoustu specializovaných metod s na míru přizpůsobenými signaturami. Nejdůležitější je ale fakt, že klientské i serverové aplikace se dají vyvíjet v různých vývojových prostředích na nejrůznějších softwarových platformách (Microsoft .NET, Oracle a Java). Manipulace se službou se nyní provádí prostřednictvím jejich softwarových součástí – objektů, tříd, funkcí nebo libovolnou jinou součástí, kterou rozhraní prostředí zveřejňuje.



Obr. 3 Ukázka webové služby, obsahující metodu AddThis a dva parametry

3.1.1 Dynamické webové knihovny

Webové služby nikdo neobjevil (alespoň ne v posledních letech), ale všichni velcí „hráči“ v aréně IT rychle adoptují a transformují ideu „softwaru volatelného z jiných aplikací“ do řešení, která vyhovují všem vývojářským platformám.

Nezávisle na tom, jak je webová služby vytvořena, kdo a pro jakou platformu ji vytvořil, zůstává v bouřlivém moři vývoje jeden pevný bod – způsob, jakým je zveřejněna; ten je standardem. Libovolnou webovou službu můžeme importovat a zahrnout do konkrétního řešení určitého výrobce, omezeného určitou platformou, musí však odpovídat užívaným standardům. Webové služby zaručují své univerzální využití, protože jsou založeny na otevřených standardech. Převodem určité funkčnosti do podoby webové služby tyto funkce nabízíte libovolné aplikaci na internetu, která umí protokoly HTTP a XML. Pochopitelně je nutná nějaká infrastruktura, která se stará o vlastní komunikaci po internetu a přenos dat. Není však nutné mít obavy – velké firmy tuto infrastrukturu začlenily přímo do softwarových vývojových prostředí.

Nový náhled na webové služby je chápán jako webové knihovny funkcí. Stejně jako dynamické knihovny Windows (DLL) jsou sdílenými kusy kódu, ze kterých mohou čerpat

nejrůznější procesy, tak i webové knihovny jsou platformě nezávislé, dynamicky připojitelné kolekce funkcí, přístupné prostřednictvím zadané adresy URL.

Jestliže porovnáme webové služby s funkcemi knihoven DLL nebo objekty COM, jsou webové služby speciálním typem systémů, orientovaných na služby. Pro komunikace nevyžadují žádné speciální protokoly. Oproti tomu třeba klient COM musí používat protokoly COM, aby mohl komunikovat se službou COM. Z toho se dá odhadnout, že COM neprostupuje web tak do hlouby jako webové služby, a rovněž trpí rutinním blokováním komunikace určitých portů na firewallech. Webové služby, založené na otevřených standardech typu HTTP a XML jsou vedoucí technologií pro univerzální komunikaci systémů.

3.1.2 Specifikace webových služeb

Webová služba má tři důležité charakteristiky. Tou první a nejdůležitější je fakt, že webová služba je na internetu dostupná prostřednictvím své adresy (URL). V případě potřeby může být její veřejná funkčnost určitým způsobem chráněna před neautorizovaným použitím pomocí běžných programovacích prostředků mezi něž paří i ověřování totožnosti a analýza uživatelů.

Za druhé, webové služby komunikují s vnějším světem prostřednictvím zpráv XML, zasílaných pomocí standardních webových protokolů. Protokol HTTP je doposud primárním komunikačním protokolem, použít však lze i jiné protokoly, například asynchronní protokoly typu SMTP nebo dokonce MSMQ.

Za třetí, webová služba je veřejně registrovaná a (což je nejdůležitější) standardním způsobem publikuje svůj popis, takže ji potencionální klienti mohou objevit a okamžitě zjistit, jak s ní pracovat. [11]

3.2 Microsoft .NET Remoting

Využití webového serveru a protokolu SOAP, které je nutné využívat u webových služeb, není pro intranetové aplikace nejvhodnější řešení. Protokol SOAP při přenosu většího množství dat zatěžuje síť. Pro rychlá intranetová řešení bychom mohli používat jednoduché soketové operace. V intranetu může ovšem existovat velké množství starších aplikací založených na objektovém modelu DCOM. V tomto modelu se volají metody objektů

spuštěných na serveru. Programovací model je stejný bez ohledu na to, zda jsou objekty používány na serveru nebo na klientovi.

Bez modelu DCOM se musíme vypořádat s porty a sokety a musíme zohledňovat cílovou platformu, protože na ní mohou být k reprezentaci dat používány jiné typy. Dále je nutné vytvořit vlastní protokol, v němž jsou zprávy posílány soketu, který nakonec volá požadované metody. Toho všeho nás model DCOM může ušetřit.

Náhradou modelu DCOM je .NET Remoting. Narozdíl od modelu DCOM může být použit rovněž v internetových řešeních, pro která není model DCOM dostatečně flexibilní a efektivní. Každou součást architektury .NET Remoting lze přizpůsobit a dokonce rozšířit. Lze ji tedy použít téměř ve všech scénářích pro vzdálenou komunikaci.

Architekturu .NET Remoting lze používat pro práci s objekty v jiné aplikační doméně bez ohledu na to, zda jsou komunikující objekty spuštěny v jednom procesu, v oddělených procesech, nebo dokonce v nezávislých systémech.

Vzdálená sestavení lze konfigurovat, aby fungovala lokálně v aplikační doméně nebo jako součást vzdálené aplikace. Je-li sestavení součástí vzdálené aplikace, používá klient objekt proxy. Objekt proxy je zástupce vzdáleného objektu v klientském procesu a klientská aplikace jej používá k volání metod. Kdykoli klient volá metodu v objektu proxy, odesílá tento objekt zprávu komunikačním kanálem vzdáleném u objektu.

Aplikace určené pro platformu .NET Framework pracují v aplikační doméně, kterou lze považovat za podproces v rámci procesu. Procesy se tradičně používali jako hranice zaručující izolaci. Aplikace spuštěná v jednom procesu nemůže používat a poškodit paměť využívanou jiným procesem. Aby mohly aplikace vzájemně komunikovat, musí mít k dispozici mechanismus komunikace mezi procesy. V prostředí .NET je aplikační doména novou bezpečnostní hranicí uvnitř procesu, protože kód MSIL je typově bezpečný a ověřitelný. Objekty v rámci jedné aplikace mohou spolu komunikovat přímo. Chtějí-li ovšem komunikovat s objekty spuštěnými v rámci jiné aplikační domény, musí použít objekty proxy.

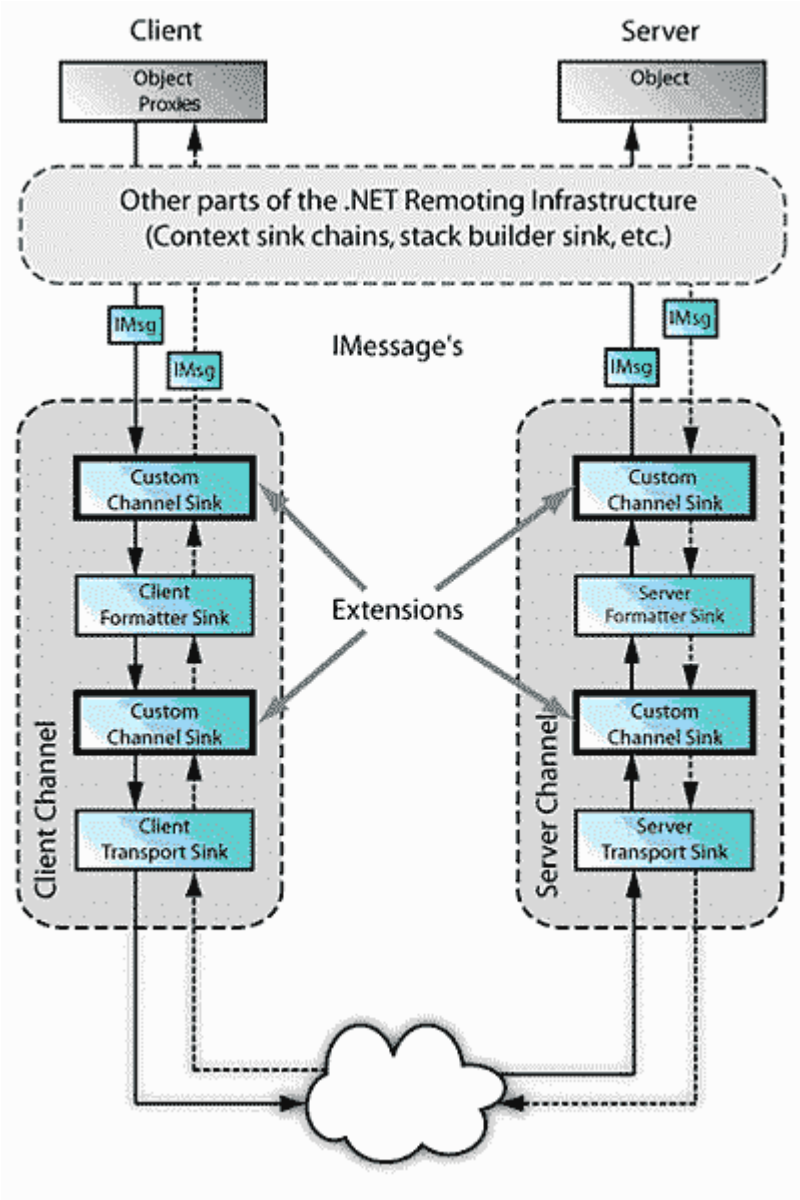
Mezi základní prvky architektury, které je třeba si vysvětlit, patří:

- **Remote object** – je objekt spuštěný na serveru. Klient nevolá metody tohoto objektu přímo, ale ke komunikaci se vzdáleným objektem používá objekt proxy. V prostředí .NET je odlišení vzdálených objektů od místních (lokálních)

jednoduché. Žádná třída odvozená od bázové třídy *MarschalByRefObject* nikdy neopustí svou aplikační doménu. Klient může volat metody vzdáleného objektu přes objekt proxy.

- **Channel** – se používá ke komunikaci mezi klientem a serverem. Rozlišujeme klientskou a serverovou část komunikačního kanálu. V platformě .NET Framework lze vytvořit dva typy komunikačních kanálů, které komunikují prostřednictvím protokolů TCP nebo HTTP. Můžeme rovněž vytvořit vlastní komunikační kanál, který ke komunikaci používá protokol jiný.
- **Messages –** jsou odesílány do komunikačního kanálu. Jsou vytvářeny pro komunikaci mezi klientem a serverem a nesou informaci o vzdáleném objektu, dále název volané metody a seznam argumentů (parametrů).
- **Formatter** – definuje způsob přenosu zpráv přes komunikační kanál a nabízí funkce pro formátování serializovaných objektů. V platfomě .NET Framework se používají formátovače SOAP a binární formátovače. Formátovač SOAP lze používat ke komunikaci s webovými službami, které nejsou založeny na platformě .NET Framework. Binární formátovače jsou mnohem rychlejší a lze je efektivně využívat v intranetovém prostředí. Samozřejmě můžeme vytvořit vlastní formátovač.
- Klient volá metody objektu **proxy**, nikoliv metody skutečného vzdáleného objektu. Existují dva typy objektů proxy: transparentní (transparent proxy) a skutečný (real proxy). Z pohledu klienta vypadá transparentní objekt proxy jako vzdálený objekt. Při práci s tímto typem objektu může klient volat metody implementované vzdálenými objekty. Metoda *Invoke()* používá příjemce zprávy, který předá zprávu komunikačnímu kanálu.
- **Message sink** – jednoduše příjemce je záchytným objektem. Takové objekty jsou na obou stranách komunikace – na straně serveru i klienta. Příjemce je přidružen ke komunikačnímu kanálu. Skutečný objekt proxy používá příjemce zpráv k předávání zpráv do komunikačního kanálu, takže příjemce může před odesláním zprávu upravit. Podle toho, kde je příjemce použit, hovoříme o zplnomocněném příjemci (envoy sink), příjemci v kontextu serveru (server context sink), o příjemci v kontextu objektu (object context sink) a dalších.

- Klient může k vytvoření vzdáleného objektu na serveru nebo k získání objektu proxy na objekt aktivovaný na serveru použít aktivátor (activator).
- Třída *RemotingConfiguration* je pomocnou třídou obsahující statické metody pro konfiguraci vzdálené infrastruktury. Lze ji použít ke čtení konfiguračních souborů, nebo k dynamické konfiguraci vzdálených objektů. [9]



Obr. 4 Ukázka komunikace technologie .NET Remoting [11]

3.3 Windows Communication Foundation

Doposud jsme mluvili o webových službách a technologii .NET Remoting. Každá z těchto technologií má své klady ovšem i větší zápory. S příchodem .NET Frameworku verze 3.0

Microsoft připravil nový jednotný framework na vytváření servisně orientovaných aplikací (SOA). Jednou ze silných stránek Windows communication Foundation (WCF) je fakt, že zjednodušuje a sjednocuje Microsoft technologie používané pro distribuované aplikace. Před příchodem Windows Communication Foundation jsme mohli využívat zejména těchto technologií:

- ASP.NET Web Service – webové služby na platformě Microsoft .NET.
- Web Service Enhancements (WSE) – doplňky pro webové služby.
- .NET Remoting – rychlost a výkonnost.
- Enterprise Services – distribuované transakce, správa životního cyklu objektů.
- Microsoft Message Queuing – spolehlivé doručování dat

	ASMX	.NET Remoting	Enterprise Services	WSE	System. Messaging	System. Net	WCF
Interoperable Web Services	X						X
Binary .NET –.NET Communication		X					X
Distributed Transactions, etc.			X				X
Support for WS-* Specifications				X			X
Queued Messaging					X		X
RESTful Communication						X	X

Obr. 5 Tabulka srovnávající technologie mezi sebou [12]

Výběr konkrétní technologie vždy záležel na požadavcích na aplikaci a jednalo se o důležitou volbu. Díky WCF již nemusíme spojovat více technologií při tvorbě spolehlivých distribuovaných aplikací, vše již je obsaženo v základu.

WCF díky implementaci platných standardů v rámci Web Services Interoperability organization (WS-I) nabízí plnou podporu jiných platforem pro přístup k datům a obecně

komunikaci se službami. Přistupovat tak ke službám vytvořeným ve WCF můžeme z Linuxového prostředí a jiných platforem.

Jednou z dalších výhod použití WCF je to, že oproti webovým službám, které vyžadují přítomnost webového serveru pro jejich interpretace. Znamená to, že mohou běžet například v konzolové aplikaci bez jakékoliv nutnosti řešení soketových operaci nebo přítomnosti webového serveru.

3.3.1 WCF Services

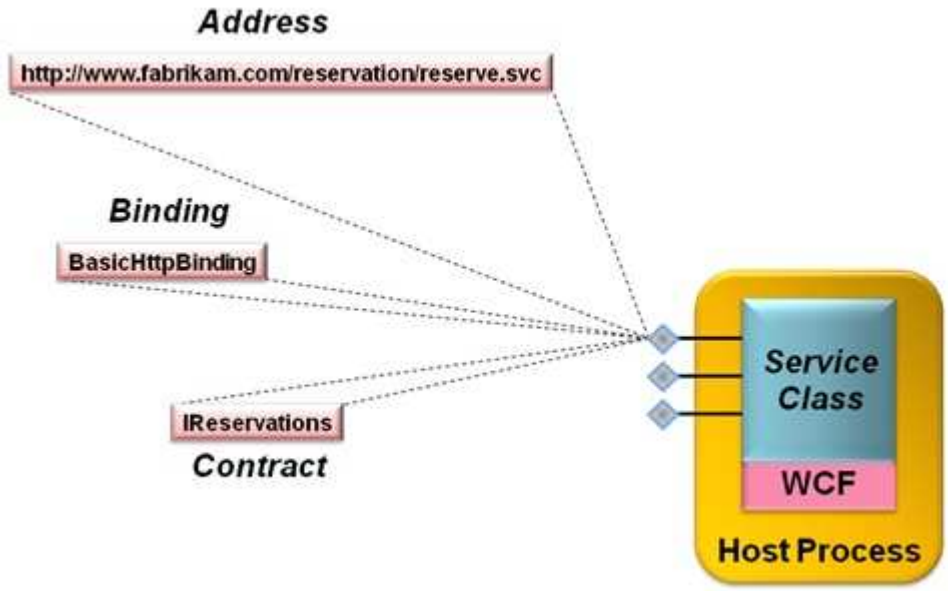
Základním kamenem WCF jsou služby. Službu si můžeme představit jako systém, se kterým komunikujeme pomocí koncových bodů (endpoint). Koncový bod si jednoznačně nadefinujeme v rámci aplikace a slouží k přijímání zpráv a odesílání odpovědí. Jako koncový bod můžeme využít klasického HTTP nebo TCP protokolu. Dle volby protokolu se mění také adresa a způsob komunikace s danou službou prostřednictvím klientské aplikace.

Koncové body jsou jednoznačně identifikovány v konfiguračním souboru aplikace a po spuštění dané aplikace dochází ke spuštění naslouchání na nadefinovaném koncovém bodě, kterým může být již zmíněný HTTP nebo TCP protokol.

V současné době můžeme využít hned několik typů koncových bodů, které můžeme využít. Jedná se zejména o:

- BasicHttpBinding – využití WS-I, která specifikuje SOAP komunikaci prostřednictvím HTTP protokolu.
- WsHttpBinding – využívá SOAP over HTTP jako výše zmíněný typ, ovšem přidává transakční podporu a doručování zpráv.
- NetTcpBinding – posílá binárně kódované SOAP zprávy prostřednictvím TCP protokolu.
- WebHttpBinding – posílá zprávy přímo přes HTTP nebo HTTPS protokol bez použití SOAP protokolu.
- NetNamedPipesBinding – umožňuje posílat zprávy mezi procesy na stejném stroji. Používá se ke komunikaci mezi WCF službami.

- NetMsmqBinding – zasílá binárně kódované SOAP zprávy přes MSMQ. Slouží pouze ke komunikaci mezi WCF službami.



Obr. 6 Ukázka principu služby s koncovým bodem [12]

4 OPTIMALIZAČNÍ ALGORITMY

4.1 Úvod do problematiky

Optimalizační algoritmy jsou mocným nástrojem pro řešení mnoha problémů inženýrské praxe. Obvykle se používají tam, kde je řešení daného problému analytickou cestou silně nevhodné či nereálné. Při vhodné implementaci mohou být použity tak, že dokonce není potřeba častého uživatelského zásahu v činnosti příslušného zařízení, kde jsou použity.

Většina problémů inženýrské praxe může být definována jako optimalizační problém jako např. nalezení optimální trajektorie robota, optimální tloušťky stěny tlakové nádoby, optimální nastavení parametrů regulátoru, optimální relace mezi fuzzy množinami atd. Jinými slovy, řešený problém lze převést na matematický problém daný vhodným funkčním předpisem, jehož optimalizace vede k nalezení argumentů tzv. účelové funkce, což je cílem optimalizace.

Příkladů lze nalézt nespočetně. Řešení takových problémů obvykle vyžaduje práci s argumenty optimalizovaných funkcí, přičemž definiční obor těchto argumentů může být různorodého charakteru jako například obor celočíselný, reálný, komplexní, diskrétní apod. Navíc se může stát (případ od případu), že pro určité subintervaly z povoleného intervalu hodnot může příslušný argument optimalizované funkce nabývat různých typů hodnot (opět celočíselný, reálný, komplexní, diskrétní apod.). Navíc v rámci optimalizace mohou být uplatněny různé penalizace a omezení nejen na dané argumenty, ale také na funkční hodnotu optimalizované funkce. Řešení takového optimalizačního problému analytickou cestou je mnohdy možné, nicméně značně komplikované a zdlouhavé.

Pro úspěšné řešení takových problémů byla v posledních dvou desetiletích vyvinuta množina velmi výkonných algoritmů, které umožňují řešit velmi složité problémy efektivním způsobem. Tato třída algoritmů má svůj specifický název a to „evoluční algoritmy“. Tyto algoritmy jsou schopny řešit velmi složité problémy tak elegantně, že se staly velmi oblíbené a používané v mnoha inženýrských oborech.

Typickým rysem pro evoluční algoritmy je, že pracují s tzv. populacemi možných řešení, jimž se říká jedinci. Tito jedinci navzájem ovlivňují svoji kvalitu na základě určitých evolučních principů v cyklech, které obvykle nesou jméno „Generace“. Cílem celého evolučního procesu je nalézt nejlepší řešení.

4.2 SOMA: Samo-Organizující se Migrační Algoritmus

SOMA je algoritmus existující od roku 1999, jehož činnost je založena na geometrických principech. Vzhledem k tomu, že pracuje s populacemi podobně jako například genetické algoritmy a výsledek po jednom evolučním cyklu (migračním kole) je totožný s genetickými algoritmy či diferenciální evolucí, lze je řadit např. mezi evoluční algoritmy navzdory faktu, že během běhu nejsou vytvářeni noví potomci, jak je tomu u jiných evolučních algoritmů. Pokud by se hledala biologická analogie, pak by se dal tento algoritmus přirovnat spíš k harémové tvorbě potomků ve stádu (jelen a laně...), než ke klasickému výběru rodičů z populace. Mnohem přesnější je však řazení mezi algoritmy matematické, jak bude později vysvětleno.

Původní myšlenka, která vedla k jeho vytvoření, spočívá v napodobení chování skupiny inteligentních jedinců, kteří kooperujíc při řešení společného problému jako je např. hledání zdroje potravy apod. SOMA od své základní verze doznal několik významných změn až do dnešní podoby, kdy se svou robustností ve smyslu nalezení globálního extrému vyrovná takovým algoritmů, jako je např. diferenciální evoluce. SOMA byl během svého postupného vylepšování publikován na různých konferencích i symposiích.

Tento algoritmus, který pracuje stejně jako ostatní evoluční algoritmy s populací jedinců, byl vyvinut na principech, které lze odpozorovat v přírodě a kterými se v sociálně-biologickém prostředí řídí inteligentní jedinci, jenž kooperují na řešení společného úkolu. Na rozdíl od ostatních evolučních algoritmů v něm totiž neprobíhá tvorba nových řešení (jedinců, potomků) filozofií křížení rodičů, ale je založena na kooperativním prohledávání (migraci) prostoru možných řešení daného problému. Vzhledem k tomu, že vlastní jádro SOMA nekopíruje již zmíněné evoluční principy, ale řídí se principy vycházejícími ze spolupráce inteligentních jedinců migrujících v prostoru možných řešení tak jako jejich biologické protějšky po krajině, byl evoluční cyklus známý jako „Generace“ zvolen název „Migrační kolo“. Příklady takového chování lze nalézt v reálném světě. Jsou to např. mravenci, včely, predátoři ve smečce hledající potravu apod.

Běh algoritmu SOMA, stejně jako ostatní evoluční algoritmy, je ovlivňován speciální množinou parametrů, které se dělí na dva druhy parametrů a to na parametry **řídící** a **ukončovací**. Řídící parametry jsou ty, které mají vliv na kvalitu běhu algoritmu (z hlediska hodnoty účelové funkce) a ukončovací jsou ty, které za předem nadefinovaných podmínek

běh algoritmu ukončují. Všechny tyto parametry musí být a priori zvoleny uživatelem ještě než započne běh algoritmu. Parametry a jejich doporučený rozsah je zobrazen v následující tabulce.

Parametr	Doporučený rozsah	Poznámka
Mass	<1.1,5>	Řídící parametr
Step	<0.11, Mass>	Řídící parametr
PRT	<0, 1>	Řídící parametr
D	dáno problémem	Počet argumentů účelové funkce
NP	<10, definuje uživatel>	Řídící parametr
Migrace	<10, definuje uživatel>	Ukončovací parametr
AcceptedError	<+- libovolný, definuje uživatel>	Ukončovací parametr

Obr. 7 Význam parametrů SOMA

4.3 Princip algoritmu SOMA

V předchozím textu bylo naznačeno, že vznik SOMA byl inspirován soutěživě-kooperativním chováním inteligentních jedinců řešících společný problém. Chování tohoto typu lze objevit prakticky kdekoliv na světě. Jako příklad může sloužit chování smečky lovících vlků, včelího úlu, termitích kolonií apod. U těchto příkladů je společným úkolem např. hledání potravy, v rámci níž jedinci spolupracují, ale i, byť nevědomky, soutěží. Ve fázi spolupráce si navzájem jednotliví jedinci sdělují, jakou kvalitu hledaného momentálně nalezli a na základě toho se snaží přizpůsobovat své chování. Ve fázi soutěžení (předchází fázi spolupráce) se každý jedinec snaží vyhrát nad ostatními – snaží se nalézt co nejlepší zdroj potravy apod. Po ukončení fáze soutěže nastane opět fáze spolupráce a jedinci si vymění informace o tom, který z nich má nejlepší zdroj potravy. Ostatní opustí své nalezené zdroje potravy a migrují (fáze soutěžení) směrem k jedinci s nejlepším zdrojem potravy a během této migrace se snaží nalézt ještě lepší zdroj. To se opakuje dokud se všichni nesejdou u nejvydatnějšího zdroje potravy. Na tomto **silně zjednodušeném** principu funguje i algoritmus SOMA.

4.4 Strategie SOMA algoritmu

V současné době existuje několik variací základního algoritmu SOMA, pro jejichž obecné označení se používá rovněž výraz „strategie“ místo „variace“ či „verze“., aby se podtrhl fakt kooperace jedinců a geometrického přesouvání populace po hyperploše, neboli

prostorem možných řešení. Všechny verze jsou navzájem prakticky plně porovnatelné ve smyslu nalezení globálního extrému. Jsou to strategie:

- „Všichni k jednomu“ (AllToOne). Tato strategie již byla popsána v předchozí sekci. Název „Všichni k jednomu“ znamená, že všichni jedinci z populace migrují k Leaderovi (samozřejmě mimo něj samotného).
- „Všichni ke všem“ (AllToAll). V této strategii neexistuje Leader. Všichni jedinci migrují ke všem ostatním tak jako ve verzi „Všichni k jednomu“ s tím rozdílem, že po dokončení migrací aktuálního jedince se daný jedinec vrací na pozici, kde byl nalezen nejlepší extrém během jeho NP-1 migračních cest vykonaných v jednom migračním kole. I když je tato strategie výpočetně náročnější, je zde vyšší pravděpodobnost, že bude nalezen globální extrém. To je způsobeno faktem, že během migrace jedince se při této strategii prohledá větší část prostoru možných řešení.
- „Adaptivně všichni ke všem“ (AllToAllAdaptive). Tato strategie je totožná se strategií „Všichni ke všem“ s tím rozdílem, že aktuálně migrující jedinec se nepřesouvá do nové pozice až po všech migracích ke všem ostatním, ale po každé, aktuálně dokončené migraci ke každému z NP-1 jedinců se přesune na lepší pozici nalezenou na této aktuální migraci. Z této tabulky pak provádí migraci k dalším zbývajícím jedincům.
- „Všichni k jednomu náhodně“ (AllToOneRand) je strategie, ve které se všichni jedinci pohybují opět k jednomu jedinci (Leaderovi), který není určen nehlubší pozicí na hyperploše, ale je pro migraci každého jedince náhodně vybrán z populace. Zde se rýsuje možná modifikace této strategie, a to taková, že jedinci nebudou vybíráni náhodně, ale podle vhodnosti tak, jak tomu je u algoritmů genetických.
- „Svazky“ (Clusters). SOMA s vytvářením svazků je úprava, která se dá použít na kteroukoliv předchozí strategii. Slovo „Svazek“ v podstatě znamená, že jedinci účastnící se migračního procesu jsou rozděleni do tzv. svazků. V každém z ni pak probíhá samostatný SOMA. Vzhledem k tomu, že se jedinci pohybují, mohou se svazky spojovat a rozpadat, což ještě více podtrhuje synergetický proces tohoto

algoritmu. Každý jedinec z populace je testován, do jakého svazku patří, podle vztahu na níže uvedeném obrázku.

$$CD \geq \sqrt{\sum_{i=1}^{\dim} \left(\frac{IND_i - CC_i}{HB_i - LB_i} \right)^2}$$

Obr. 8 *Vztah vyjadřující do jakého svazku jedinec patří*

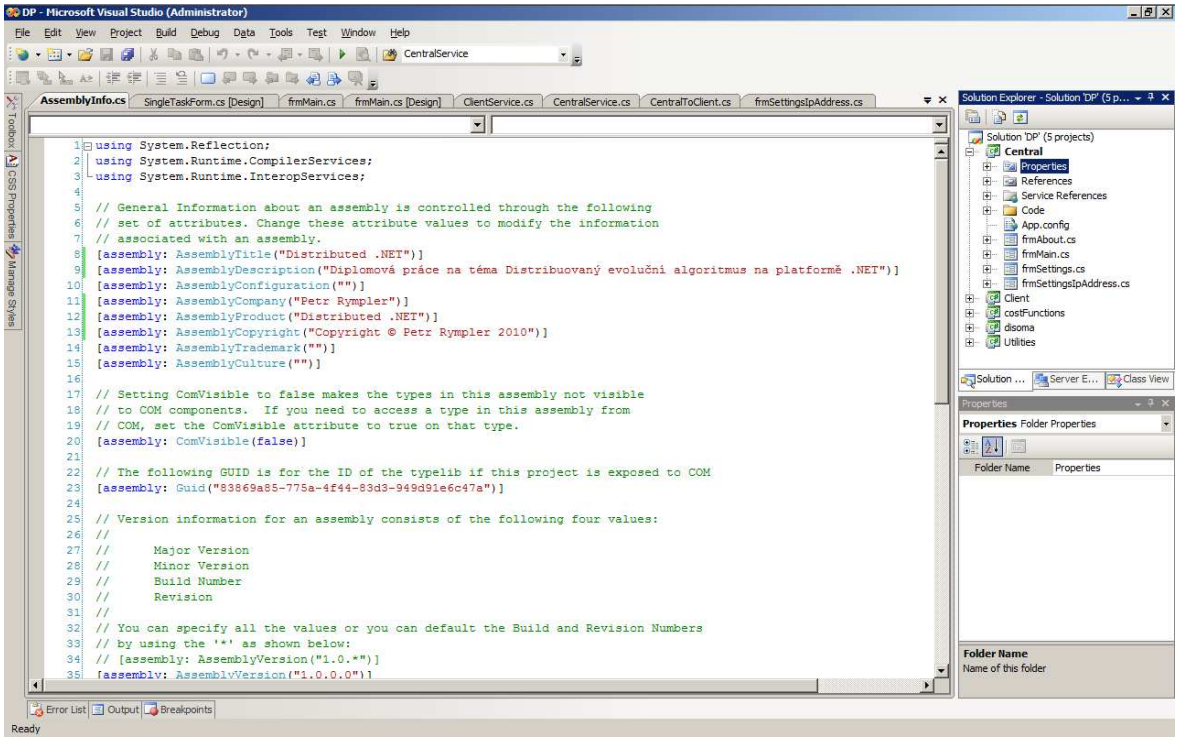
Parametr INDi je i-tý parametr jedince, CCi je i-tý parametr Leadera (centrum svazku), HBi a LBi jsou povolené hranice jednotlivých parametrů. CD je velikost svazku daná uživatelem. Výsledkem je tedy vytvoření svazků obsahujících jedince populace. Pokud se stane, že nějaký jedinec je příliš daleko od ostatních a tudíž nemůže být zahrnut do již existujících svazků, je svazkem sám sobě a migruje ke všem ostatním jako v AllToALl. [13]

II. PRAKTICKÁ ČÁST

5 POPIS ŘEŠENÍ A VYUŽITÍ NÁSTROJŮ

5.1 Využití vývojového prostředí Visual Studio .NET

Cele řešení je vytvořeno na platformě Microsoft .NET za použití vývojového nástroje Microsoft Visual Studio .NET 2008. Toto vývojové prostředí nabízí vhodné nástroje pro úplné začátečníky v programování, jednotlivé profesionály i týmy hledající prostředí podporující moderní metodiky a principy životního cyklu vývoje softwarových aplikací.



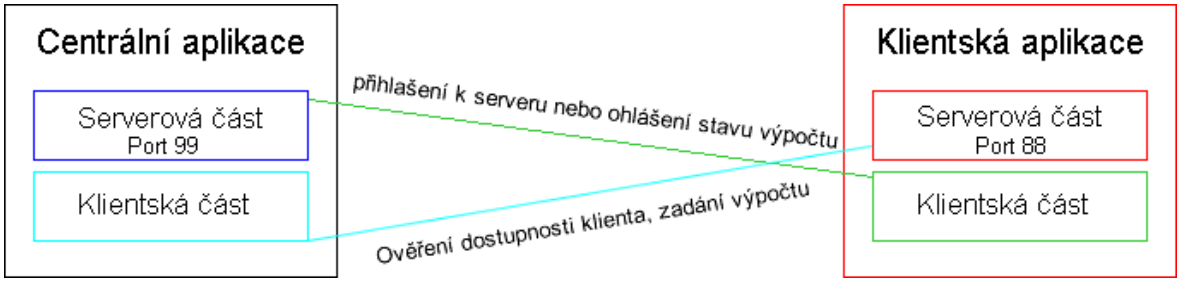
Obr. 9 Ukázka vývojového prostředí Visual Studio .NET

5.2 Využití technologie WCF

Celé řešení je postaveno na komunikaci mezi centrální aplikaci a klientskými službami, které komunikují pomocí WCF služeb na obou stranách.

Centrální aplikace je aplikace, která obsluhuje klienty a umožňuje na nich spouštění zadaných úloh. Tato část aplikace je řešena pomocí grafického Windows Form rozhraní. O vlastnostech a vzhledu rozhraní centrální i klientské aplikace se budeme věnovat v dalších odstavcích.

Samotný způsob komunikace pomocí WCF služeb bychom mohli shrnout v následujícím obrázku.



Obr. 10 Způsob komunikace Centrální a klientské aplikace

6 ZPŮSOB KOMUNIKACE A DISTRIBUCE EVOLUČNÍHO ALGORITMU

K distribuci a komunikaci evolučního algoritmu je využito technologie Windows Communication Foundation (WCF), jejíž popis byl zmíněn v teoretické části.

6.1 Využití Windows Communication Foundation

Pro komunikaci se využívá klientská a serverová část WCF, která je umístěna v centrální i klientské aplikaci. Klient i centrální aplikace komunikují pomocí *BasicHttpBinding*, který popisuje způsob komunikace endpointů jednotlivých služeb.

6.2 Zabezpečení Windows Communication Foundation

U tohoto typu bindingu není oproti jiným typům jako například *NetTcpBinding*, *WSHttpBinding* potřeba definovat na obou stanicích stejné uživatelské účty pro přístup ke službám. WCF si totiž ve výchozím nastavení provádí autentizaci pomocí cílových a zdrojových účtů a v případě nedodání správného uživatelského účtu spolu služby nebudou komunikovat.

V tuto chvíli WCF podporuje několik typů ověřování:

- Anonymní volání
- Windows credential – zabezpečení pomocí uživatelských účtů.
- CardSpace – zabezpečení pomocí technologie CardSpace.
- Certifikáty – zabezpečení pomocí klientských a serverových certifikátů.
- Username and password – zabezpečení pomocí přihlašovacího jména a hesla.

My si nyní na konkrétních ukázkách ukážeme ty nejpoužívanější.

6.2.1 Zabezpečení pomocí uživatelských účtů

Jedná se o výchozí zabezpečení komunikace mezi klientem a serverem ve WCF. Tento typ zabezpečení je výchozí u všech bindingů kromě *BasicHttpBinding* a využívá integrované ověření Windows.

Jestliže jsou klientská i serverová část umístěny ve Windows doméně, musejí se k síti přihlásit pomocí uživatelského jména a hesla. Tyto uživatelské přihlašovací údaje jsou využity k ověření identity a v případě, že si uživatelské účty neodpovídají nebo nemají dostatečné oprávnění, dojde hned v počátcích komunikace k okamžitému přerušení spojení mezi klientem a serverem.

```
<security mode="Transport">
   <transport clientCredentialType="None" />
</security>
```

Obr. 11 Ukázka výchozího zabezpečení WCF na straně klienta

6.2.2 Zabezpečení pomocí klientských certifikátů

Další z možností zabezpečení komunikace je zabezpečení pomocí klientských certifikátů. Při tomto způsobu zabezpečení se využívá certifikát, který je nainstalován (přidána do Certificate Store) na klientském i serverovém PC.

Jako první je vygenerován společný certifikát, který budeme používat na straně klienta i na straně serveru. Tento certifikát po vygenerování musíme přenést a nainstalovat na stranu klienta i serveru do Certificate Store (Trusted Root Certification Authorities).

```
<security mode="Transport">
   <transport clientCredentialType="None" />
</security>
```

Obr. 12 Ukázka zabezpečení pro využití certifikátů na straně klienta

Dále je třeba nastavit využívání certifikátu na straně serverové aplikace.

```
<serviceCertificate
      x509FindType="FindBySubjectName"
      findValue="computer.mydomain.com"
         storeLocation="LocalMachine"
         storeName="My" />
</serviceCredentials>
```

Obr. 13 Ukázka zabezpečení pomocí certifikátu na straně serveru.

6.2.3 Anonymní volání

Při tomto způsobu volání není klient na straně serveru nijak ověřován a můžeme volat jednotlivé služby bez nutnosti přihlašování. Toto zabezpečení však není doporučováno pro

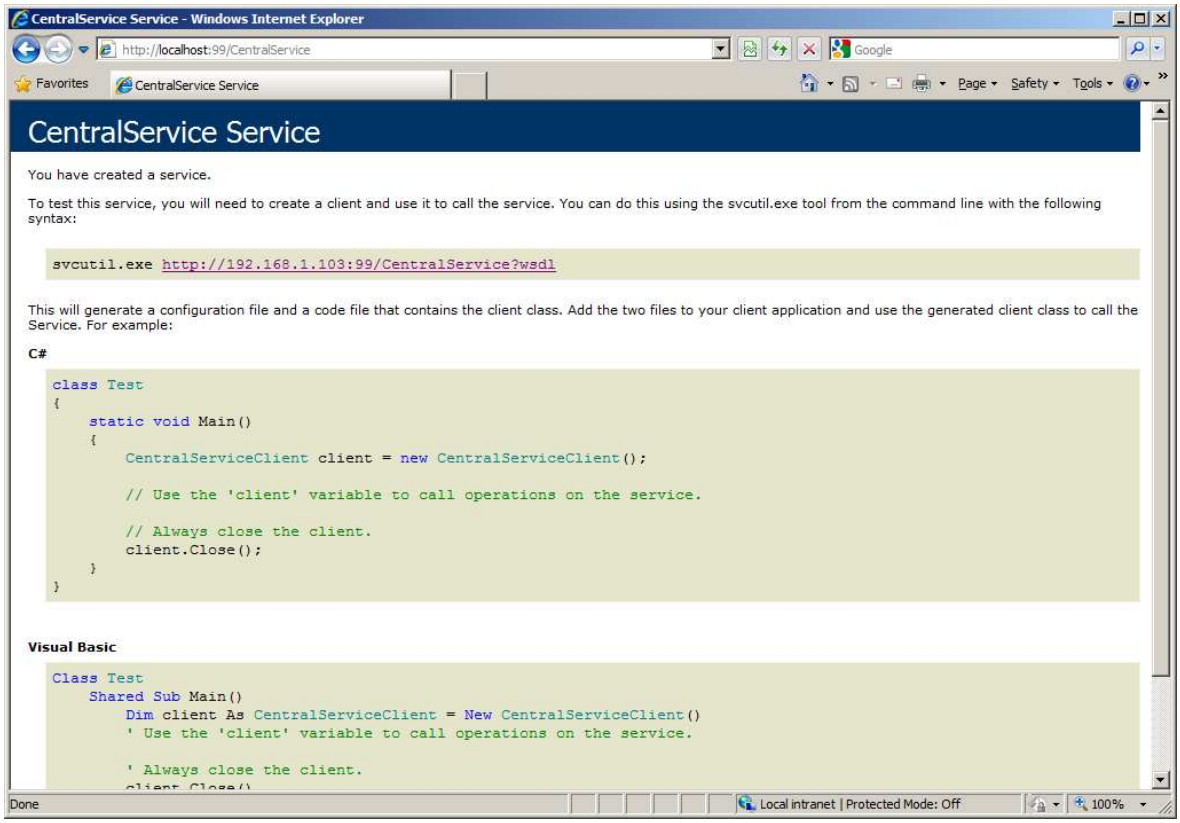
internetové nebo extranetové aplikace. V rámci testování a intranetu se toto řešení může využít za předpokladu využití HTTPS šifrovaného kanálu apod.

```
<security mode="None">
   <transport clientCredentialType="None" />
</security>
```

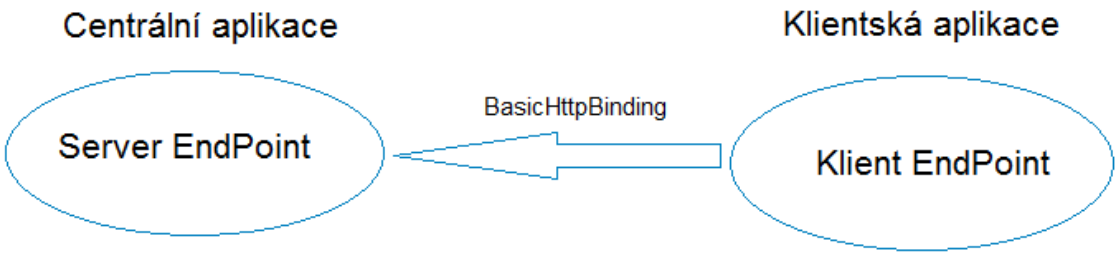
Obr. 14 Ukázka nastavení anonymního volání na straně klienta.

6.3 Komunikace s centrální aplikací

Centrální aplikace je řešena formou HTTP služby využívající *BasicHttpBinding*, která poslouchá na portu 99. Na tomto portu je realizován integrovaný webový server WCF služby, který vyřizuje požadavky. Celá adresa služby je ve tvaru http://<ipadresa>:99/CentralService a ve velkém míře se podobá volání webové služby. To znamená, vždy přidat referenci na službu a následně pracujeme s objektem (instancí třídy), který zprostředkovává komunikaci mezi oběmi EndPointy.



Obr. 15 Ukázka služby CentralService



Obr. 16 Ukázka komunikace s centrální aplikací

Základem toho, aby klient vůbec mohl komunikovat se serverovou (v tomto případě centrální) aplikací (službou) je nutnost definice služby v konfiguračním souboru aplikace (App.config soubor). Celá konfigurace se provádí změnou patřičných konfiguračních voleb ve formátu XML.

Serverovou část zajišťuje definice Services, která definuje seznam služeb (services), koncových bodů (EndPoints) a typ Bindings. Za zmínku stojí uvedení adresy pomocí elementu <baseAddress>, který definuje adresu, na níž bude naše služba naslouchat požadavkům. U *BasicHttpBinding* dojde k vytvoření HTTP serveru na daném portu, který bude oproti *NetTcpBinging* naslouchat na všech IP adresách, které má daný počítač k dispozici.

```
<!--SERVER-->
<services>
  <service
  behaviorConfiguration="PR.DP.WcfCentralService.CentralService1Behavior"
  name="PR.DP.WcfCentralService.CentralService">
  <endpoint address=""
            bindingConfiguration=""
            contract="PR.DP.WcfCentralService.ICentralService">
      <identity>
        <dns value="localhost" />
      </identity>
  </endpoint>
  <endpoint
    address="mex"
    binding="mexHttpBinding"
    contract="IMetadataExchange" />
      <host>
        <baseAddresses>
           <add baseAddress="http://192.168.1.103:99/CentralService" />
        </baseAddresses>
      </host>
   </service>
</services>
<behaviors>
  <serviceBehaviors>
    <behavior name="PR.DP.WcfCentralService.CentralService1Behavior">
      <serviceMetadata httpGetEnabled="True"/>
      <serviceDebug includeExceptionDetailInFaults="True" />
```

```
    </behavior>
  </serviceBehaviors>
</behaviors>
<!--END SERVER-->
```

Obr. 17 Ukázka konfiguračního serveru na straně centrální aplikace

Nyní máme připravenu službu, která naslouchá na HTTP portu 99 a klientský počítač se k ní může připojit velice jednoduše. Stačí si přidat službu jako referenci a vytvořit instanci objektu zajišťujícího komunikaci.

```
1. // Vytvoření instance třídy zajišťující komunikaci
2. // s centrální aplikací
3. CentralServiceClient centralService = new CentralServiceClient();
4.
5. // Zavolání ukázkové metody, která označí klienta jako online
6. centralService.ClientIsOnline(clientip);
```

Po přidání reference na tuto službu se v konfiguračním souboru aplikace automaticky definuje klientská část, která určuje adresu serveru a další vlastnosti připojení. Můžeme zde definovat timeouty pro odeslání požadavku, pro příjem požadavku, nastavení velikosti požadavku a další nastavení.

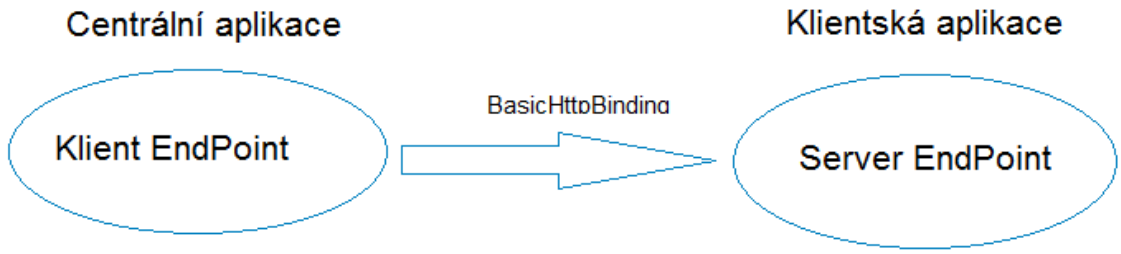
```
<!--CLIENT-->
<bindings>
  <basicHttpBinding>
    <binding
       name="BasicHttpBinding_IClientService" closeTimeout="00:01:00"
       openTimeout="00:01:00" receiveTimeout="00:10:00"
       sendTimeout="00:01:00"
       allowCookies="false" bypassProxyOnLocal="false"
       hostNameComparisonMode="StrongWildcard"
       maxBufferSize="65536" maxBufferPoolSize="524288"
       maxReceivedMessageSize="65536"
       messageEncoding="Text" textEncoding="utf-8"
       transferMode="Streamed" useDefaultWebProxy="true">
          <readerQuotas maxDepth="32"
             maxStringContentLength="8192"
             maxArrayLength="16384"
             maxBytesPerRead="4096"
             maxNameTableCharCount="16384" />
          <security mode="None">
            <transport
               clientCredentialType="None"
               proxyCredentialType="None" realm="">
            </transport>
            <message
               clientCredentialType="UserName"
               algorithmSuite="Default" />
          </security>
        </binding>
      </basicHttpBinding>
    </bindings>
    <client>
      <endpoint
          address="http://192.168.1.103:88/ClientService"
          binding="basicHttpBinding"
          bindingConfiguration="BasicHttpBinding_IClientService"
```

```
          contract="ClientService.IClientService"
          name="BasicHttpBinding_IClientService" />
    </client>
    <!--END CLIENT-->
```

Obr. 18 Ukázka konfiguračního souboru na straně server – klientská část

6.4 Komunikace s klientskou aplikací

Komunikace s klientskou aplikací pracuje analogicky ke komunikaci s centrální aplikací. Každý klient mu zaregistrovánu vlastní službu, která naslouchá na portu 88. Jedná se opět o *BasicHttpBinding*, který má na straně klienta serverovou část jenž provádí příkazy z centrální aplikace.



Obr. 19 Ukázka komunikace s klientskou aplikací

Služba je obdobně definována v konfiguračním souboru aplikace.

```
<!--SERVER-->
<services>
  <service
  behaviorConfiguration="PR.DP.WcfClientService.ClientService1Behavior"
  name="PR.DP.WcfClientService.ClientService">
  <endpoint
    address=""
    binding="basicHttpBinding"
    bindingConfiguration=""
    contract="PR.DP.WcfClientService.IClientService">
      <identity>
        <dns value="localhost" />
      </identity>
  </endpoint>
  <endpoint
    address="mex"
    binding="mexHttpBinding"
    contract="IMetadataExchange" />
      <host>
        <baseAddresses>
           <add baseAddress="http://192.168.1.103:88/ClientService" />
        </baseAddresses>
      </host>
   </service>
</services>
<behaviors>
```

```
  <serviceBehaviors>
    <behavior name="PR.DP.WcfClientService.ClientService1Behavior">
      <serviceMetadata httpGetEnabled="True"/>
      <serviceDebug includeExceptionDetailInFaults="True" />
    </behavior>
  </serviceBehaviors>
</behaviors>
<!--END SERVER-->
```

Obr. 20 Konfigurace serveru na straně klienta

7 DEFINICE OBJEKTŮ SLOUŽÍCÍCH KE KOMUNIKACI

Samotná způsob komunikace a pouhá definice koncových bodů (EndPoint) by byla k ničemu bez definice objektů služeb, které zajišťují volání metod a distribuci objektu mezi centrální aplikací a klientskou aplikací.

WCF služby, které budou přístupné musí implementovat vlastní rozhraní. Toto rozhraní obsahuje seznam metod, které třída implementující toto rozhraní musí obsahovat a také jejich parametry a návratové hodnoty.

```
1.  namespace PR.DP.WcfCentralService
2.  {
3.     [ServiceContract]
4.     public interface ICentralService
5.     {
6.         [OperationContract]
7.         void ClientIsOnline(string ip);
8.
9.         [OperationContract]
10.        void ReturnResults(string ip, double[] data);
11.    }
12.  }
```

7.1 Service contract

Service contact popisuje operace, které daná služba může vykonávat. Nový Service contract definujeme přidáním atributu *[ServiceContract]* před definici třídy nebo rozhraní. Jednotlivé operace služby můžeme definovat přidáním atributu *[OperationContract]*, kdy takto označené metody jsou viditelné a mohou být volání z klienta.

7.2 Data contract

Definuje strukturu dat, kterou můžeme přenášet oproti základním datovým typům jako int, string apod. Abychom mohli takovou datovou strukturu přenášet, musíme ji označit atributem *[DataContract]*, kdy jednotlivé metody nebo vlastnosti uvnitř této datové struktury musíme označit atributem *[DataMember]*.

```
1.     [DataContract]
2.     public class ComputeRequest
3.     {
4.         private string type;
5.         private double[] data;
6.
7.         [DataMember]
8.         public string Type
9.         {
10.            get { return this.type; }
11.            set { this.type = value; }
```

```
12.         }
13.
14.         [DataMember]
15.         public double[] Data
16.         {
17.             get { return this.data; }
18.             set { this.data = value; }
19.         }
20.     }
```

7.3 Implementace rozhraní

Jakmile máme připraveno rozhraní a datové struktury pro přenos dat, musíme vytvořit třídu implementující rozhraní. V aplikaci jsou takové třídy dvě a jedná se o třídu *CentralService* a *ClientService*.

7.3.1 Třída CentralService

Třída *CentralService* obsahuje definice jednotlivých metod, které obsahuje rozhraní *ICentralService*. Tato třída obsahuje několik metod, které jsou jednoduché a určují činnost aplikace:

- Metoda *ClientIsOnline(string ip)* – umožňuje přihlásit se k centrální aplikaci jako počítač k dispozici. Centrální aplikace má vždy přehled o aktuálně přihlášených počítačích, na které může být provedena distribuce zadání provedena.

7.3.2 Třída ClientService

Třída *ClientService* implementuje rozhraní *IClientService*, jenž definuje seznam metod, které může centrální aplikace volat:

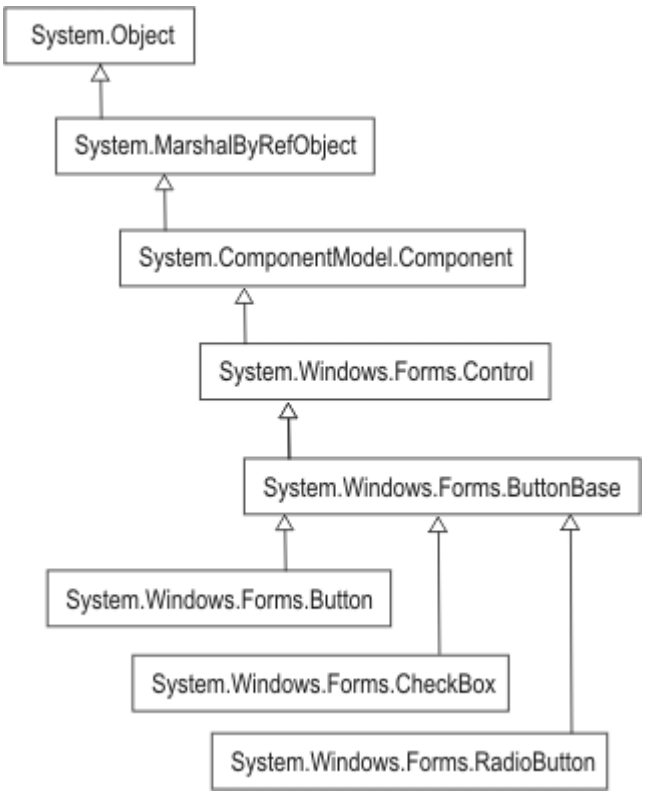
- Metoda *UpdateClientIpAddress(string ip)*– umožňuje nastavit klientovi novou IP adresu server, na kterou se klient snaží po spuštění připojit.

8 ŘEŠENÍ CENTRÁLNÍ – ŘÍDÍCÍ APLIKACE

Centrální aplikace je řešena jako Windows Forms aplikace skládající se z několika samostatných formulářů, které spolu komunikují.

8.1 Windows Forms aplikace

Platforma Microsoft .NET nabízí možnost tvorby nejen webových služeb a systémových služeb, ale také prostředí pro tvorbu grafických desktopových aplikací. Dříve před platformou Microsoft .NET bylo třeba využívat Windows API s využitím programovacího jazyku C nebo C++. Takto vyvíjené aplikace však nevynikali zrovna svojí pružností a jednoduchostí, což vedlo ke složitému kódu a mnoha chybám. Částečné ulehčení přišlo s přístupem knihovny MFC, avšak stále to nebylo to pravé ořechové. S příchodem .NET platformy bylo vytvořeno nové rozhraní, které komunikuje přímo s vrstvou Win32 API a umožňuje jednoduchou tvorbu grafických aplikací a formulářů. Díky vlastnostem .NET Frameworku můžeme formulářové aplikace vyvíjet a jakémkoliv jazyku, který je dostupný platformě .NET.



Obr. 21 Hierarchie třídy u ovládacího prvku RadioButton [14]

Třídy pro práci s grafickými objekty vycházejí z jmenného prostoru *System.Windows.Forms*, který je velmi rozsáhlý a obsahuje bázové třídy jednotlivých grafických objektů jako například tlačítka (Button), vstupní pole (TextBox), kontejnery (Panel) a mnohé další.

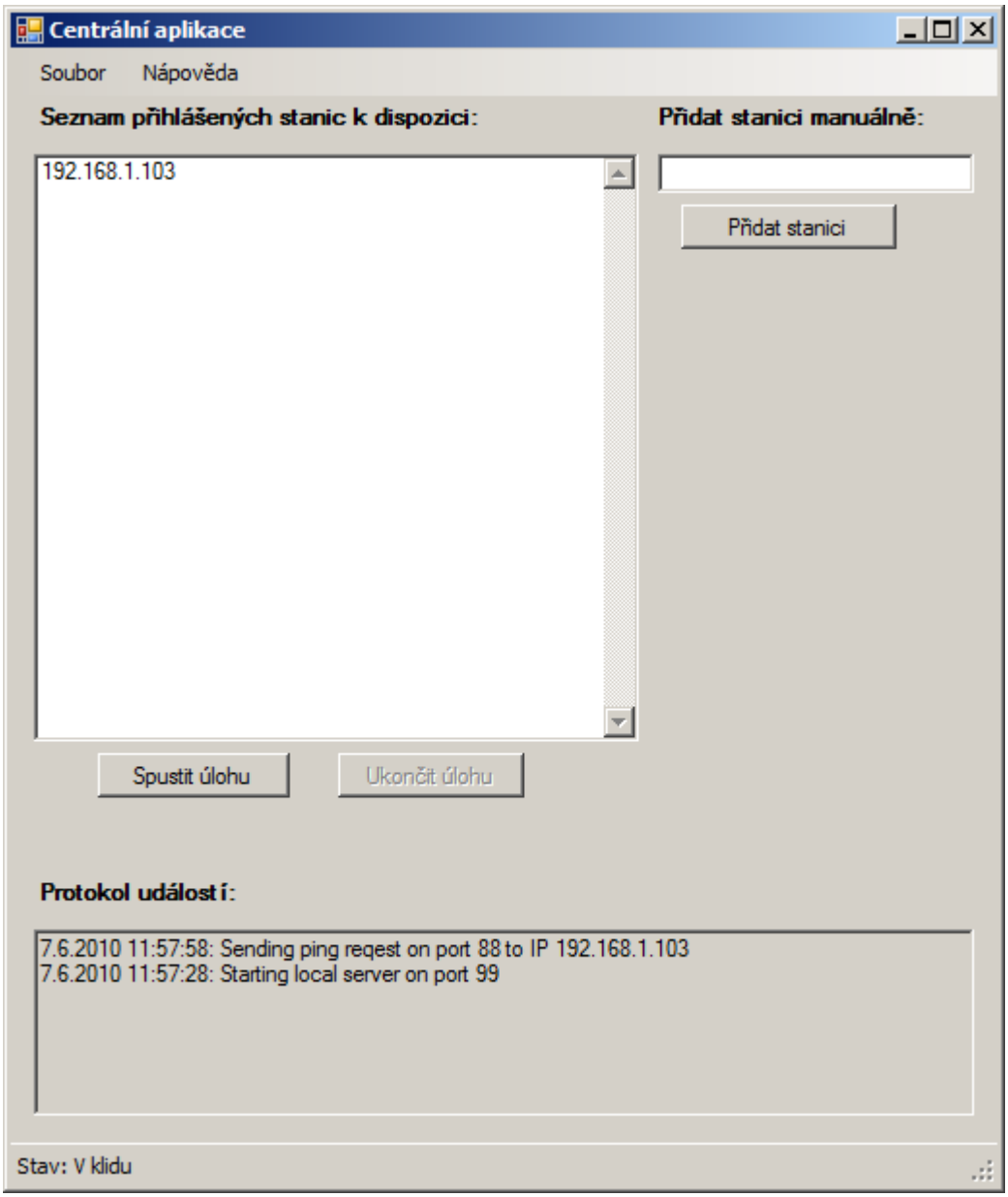
Mezi nejznámější ovládací prvky ve Windows Forms aplikacích patří:

- Tlačítko - instance třídy Button.
- Zaškrtávací políčko - instance třídy CheckBox.
- Přepínač - instance třídy RadioButton.
- Popisek - instance třídy Label.
- Hypertextový popisek - instance třídy LinkLabel.
- Textové pole - instance třídy TextBox.
- Skupinový rámeček - instance třídy GroupBox.
- Panel - instance třídy Panel.
- Seznam - instance třídy ListBox.
- Zaškrtávací seznam - instance třídy CheckedListBox.
- Rozbalovací seznam - instance třídy ComboBox.
- Vodorovný posuvník - instance třídy HScrollBar.
- Svislý posuvník - instance třídy VScrollBar.
- Číselník - instance třídy NumericUpDown.
- Posuvný jezdec - instance třídy TrackBar.
- Ukazatel průběhu - instance třídy ProgressBar. [14]

8.2 Hlavní formulář aplikace

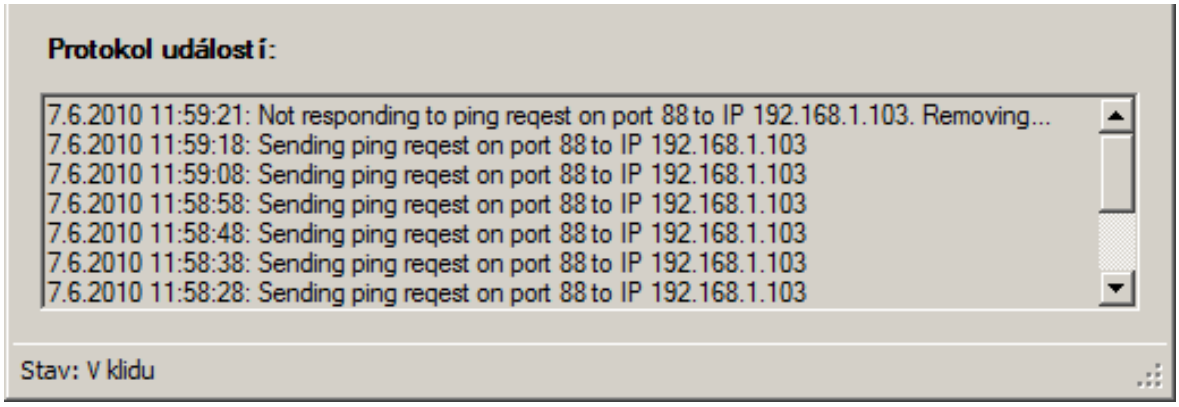
Hlavní aplikace (centrální aplikace) je tvořena formulářem, který obsahuje menu **Soubor**, s jehož pomocí se můžeme přepnout do formuláře Nastavení aplikace. S pomocí menu Soubor můžeme také aplikaci ukončit a uvolnit systémové prostředky. Kliknutím na menu Nápověda se zobrazí formulář Nápověda, obsahující informace o verzi a popisu aplikace.

Samotné okno se skládá z několika sekcí, které budou v následujících odstavcích popsány.



Obr. 22 Ukázka rozhraní centrální aplikace

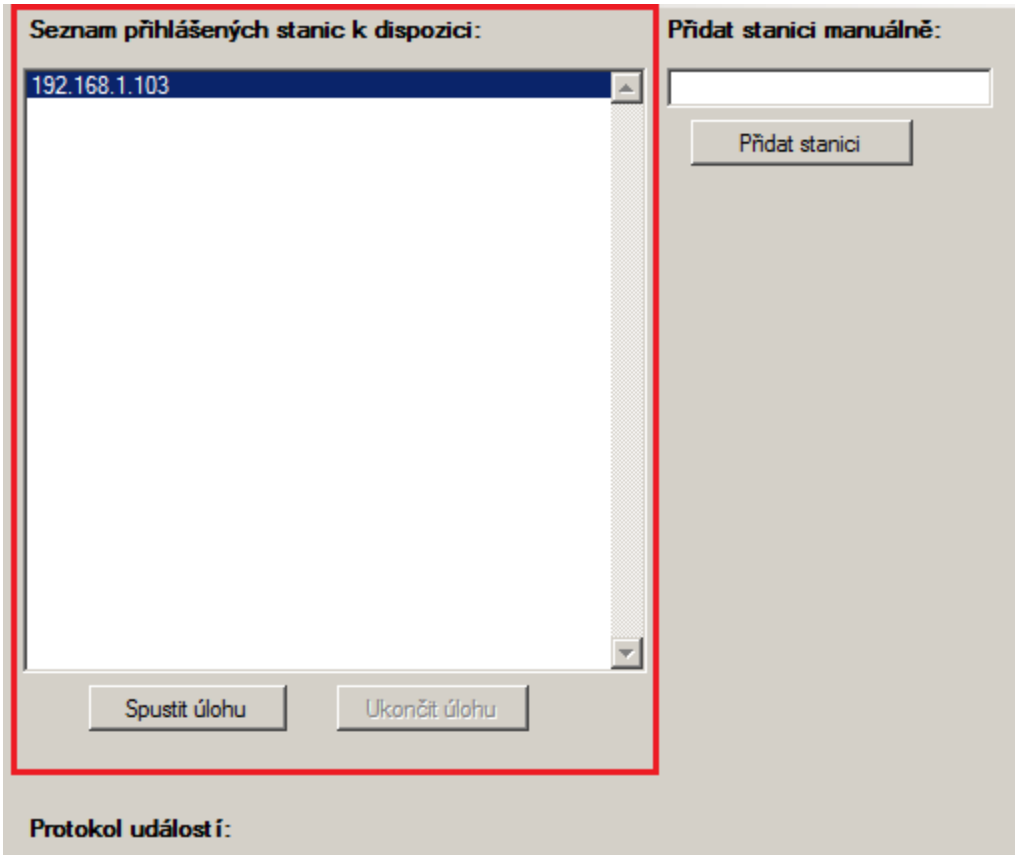
V dolní části aplikace je uveden protokol událostí, který zobrazuje datum, čas a text události, která nastala. Zobrazují se zde informativní události, o kterých je dobré vědět.



Obr. 23 Ukázka protokolu událostí

V levé části aplikace je umístěn ovládací prvek *ListBox*, který vždy obsahuje seznam aktuálně připojených klientů, se kterými můžeme pracovat.

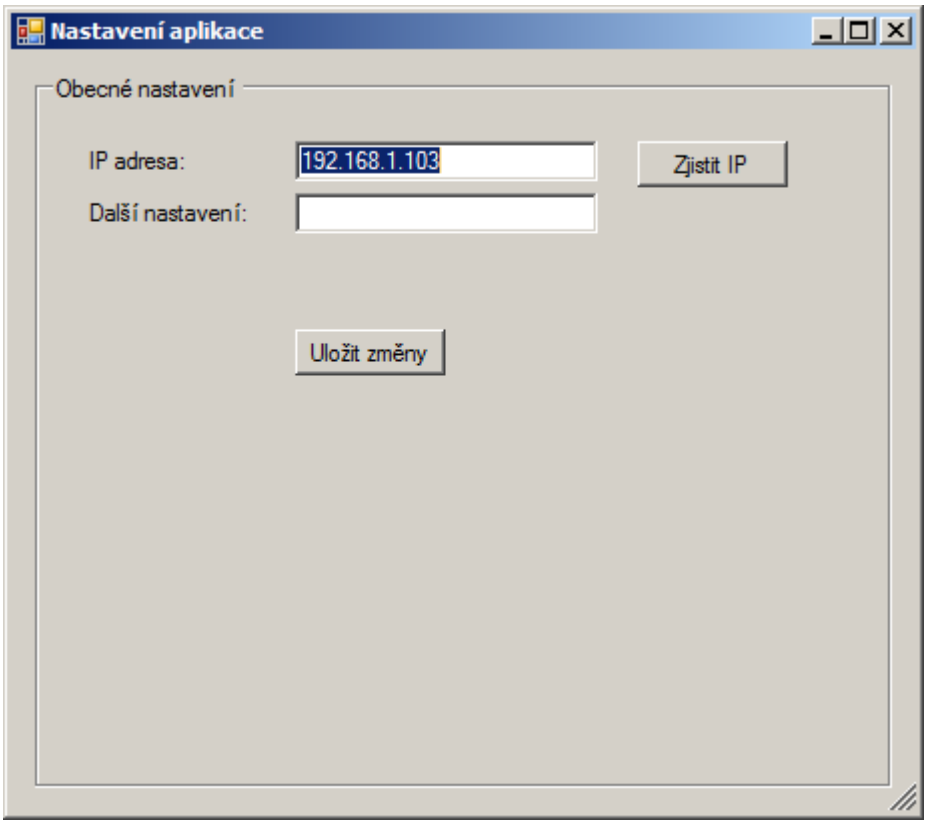
Centrální aplikace provádí v předem nastavený časový interval pomocí ovládacího prvku *Timer* kontrolu všech připojených klientů voláním serverové části klientské stanice. Pokud klient neodpovídá (z důvodu vypnutí), dojde k jeho odebrání z tohoto ovládacího prvku.



Obr. 24 Ukázka rozhraní umožňujícího spuštění úlohy

8.3 Formulář Nastavení aplikace

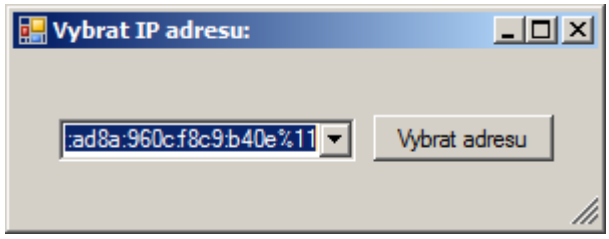
Tento formulář umožňuje změnu IP adresy centrální aplikace, ke které může dojít při přesunu centrální aplikace do jiné sítě. Před změnou adresy po kliknutí na tlačítko **Uložit změny** se provede zavolání metody *UpdateClientIpAddress(string ip)*, která zajistí změnu IP na všech připojených klientských aplikacích.



Obr. 25 Ukázka formuláře Nastavení aplikace

8.4 Formulář Vybrat IP adresu

Tento formulář slouží k vybrání jedné z IP adres, které jsou registrovány na lokálním počítači. Po kliknutí na tlačítko **Vybrat adresu** se adresa přenese do předchozího formuláře Nastavení.



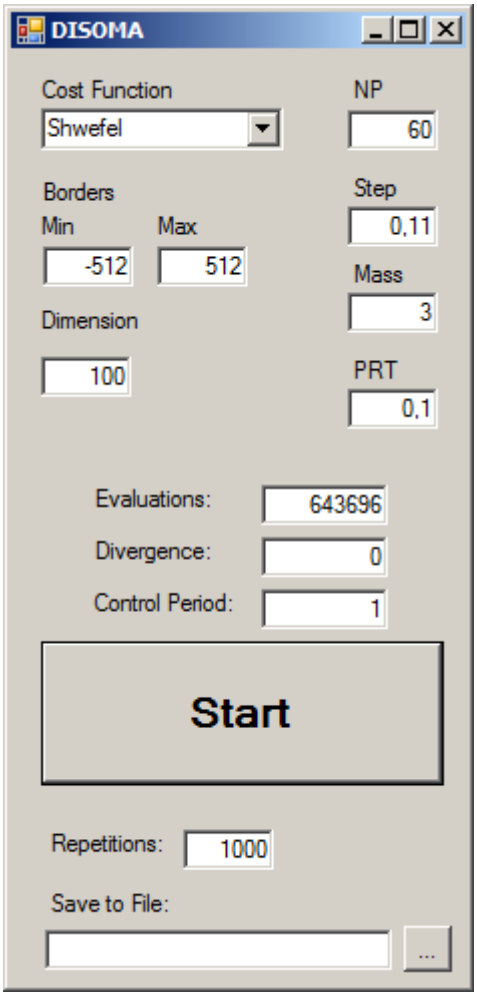
Obr. 26 Ukázka fomuláře Vybrat IP adresu

8.5 Formulář O aplikaci

Jedná se o klasický (generický) formulář (about box), který můžeme do své aplikace umístit. Informace o verzi, názvu aplikace a popisu jsou čerpány ze soubor *AssemblyInfo.cs*, který je většinou součástí každé aplikace.

Obr. 27 Formulář O aplikaci

Po kliknutí na tlačítko **Start** v hlavním okně centrální aplikace dojde k otevření okna pro zadání hodnot a výběru funkce, které se následně budou distribuovat na klienty. Tento formulář obsahuje pole pro zadání několika parametrů, které jsou uvedeny v teoretické části.



Obr. 28 Formulář pro spuštění výpočtu

9 ŘEŠENÍ KLIENTSKÉ APLIKACE

Klientská aplikace je řešena v prostředí Microsoft .NET Framework jako systémová služba, kterou lze spustit automaticky při spuštění operačního systému. Po spuštění se aplikace snaží připojit na IP adresu serveru, kterou má uloženu v konfiguračním souboru spolu se sekcemi definujícími jednotlivé koncové body (end points). Pokud se klientská aplikace nemůže připojit k centrálnímu počítači, využije nadefinované HTTP adresy, jenž je uložena v konfiguračním souboru, odkud se snaží stáhnout aktuální IP adresu. Tato skutečnost řeší to, že centrální aplikace si změní IP bez možnosti oznámení změny jednotlivým klientům.

9.1 Systémové služby

Takové aplikace mohou vykonávat svoji činnost bez zásahu přihlášeného uživatele. Službu systému Windows můžeme konfigurovat pro spuštění prostřednictvím specifického uživatelského účtu nebo prostřednictvím účtu systémového uživatele - což je uživatelský účet, která má dokonce větší oprávnění než správce systému.

Pro chod služby systému Windows jsou nezbytné tři typy programů:

- Aplikace služby
- Program řízení služby
- Konfigurační program služby

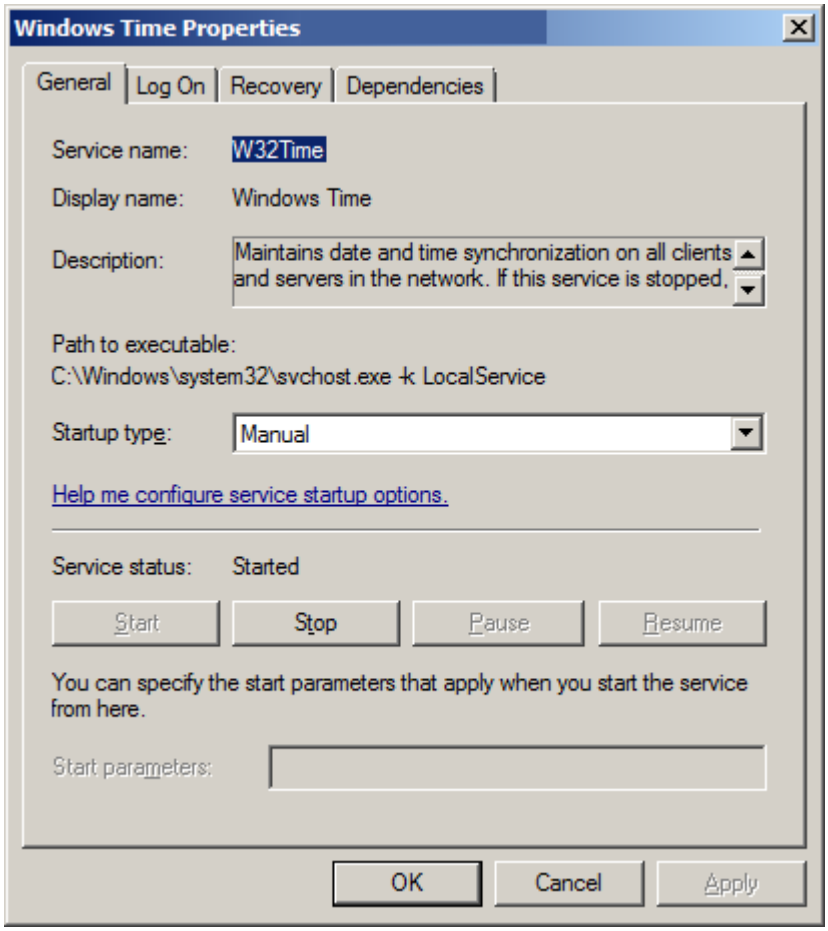
Aplikace služby poskytuje skutečné funkce, které potřebujeme. Pomocí programu řízení služby můžeme službě posílat požadavky, jako jsou příkazy: spustit, zastavit, pozastavit a pokračovat. Nakonec potřebujeme rovněž konfigurační program služby. Tento program umožňuje instalaci aplikace služby. Aplikace je zkopírována do souborového systému, zapsána v registru a konfigurována jako služba. Zatímco komponenty .NET lze instalovat pouhým kopírováním, protože nevyžadují přítomnost záznamů v registru, instalace služby vyžaduje konfiguraci v systémovém registru. Konfigurační program služby lze používat rovněž k pozdějším modifikacím konfigurace služby

9.1.1 Aplikace služby

Aplikace služby implementuje funkci služby a vyžaduje tři součásti:

- Hlavní funkci
- Hlavní funkci služby
- Obslužnou funkci (handler)

Hlavní funkce musí registrovat hlavní funkce všech poskytovaných služeb. Aplikace služby může prostřednictvím jednoho programu poskytovat mnoho služeb. Správce služeb nyní volá funkce všech spuštěných služeb. Hlavní funkce obsahuje skutečnou funkci služby a jednou z jejích důležitých úloh je registrace obslužné funkce u správce služeb.



Obr. 29 Ukázka služby Windows Time v systému Windows Server

Obslužná funkce (handler function) je třetí součástí aplikace služby. Musí reagovat na události správce služeb, stejně jako na službě samotné. Operační systém obsahuje mnoho programů řízení služeb.

9.1.2 Program řízení služby

Jak již název napovídá, je program řízení služby určen k řízení chodu služby. Chceme-li chod služby zastavit, pozastavit nebo obnovit, můžeme službě poslat řídící kódy a obslužná funkce by na ně měla odpovídajícím způsobem reagovat. Můžeme vyslat dotaz na aktuální stav služby nebo implementovat vlastní obslužnou funkci, která bude reagovat na vlastní řídící kódy.

9.1.3 Konfigurační program služby

K instalaci služby nemůžeme využít program *xcopy*, protože služby je třeba konfigurovat v systémovém registru. Můžeme nastavit typ spuštění služby jako automatický nebo ruční, či spouštění služby zakázat. Musíme konfigurovat uživatele aplikace služby a závislosti na službě, nebo naopak závislost naší služby na službách jiných. Před spuštěním služby musí tedy být v systému už spuštěny určité další služby. Všechny tyto konfigurace zajišťuje konfigurační program služby. Instalační program může konfigurační program služby použít ke konfiguraci služby. Tento program lze ovšem využít i později k úpravám nastavení konfiguračních parametrů. [9]

ZÁVĚR

Účelem této diplomové práce bylo vytvoření systému pro distribuci a komunikaci výpočtů mezi zvolenou centrální aplikací a množinou klientských aplikací. Celé řešení se skládá ze dvou částí, kdy první částí je centrální aplikace, která rozesílá a ovládá množinu klientských aplikací (stanic). Klientská aplikace naslouchá na nadefinovaném portu a čeká na požadavky centrální aplikace. Obě části jsou naprogramovány na platformě Microsoft .NET s využitím moderní technologie Windows Communication Foundation a Windows Forms (pouze centrální část). Ani jedna aplikace nevyužívá databázový server, což ji do jisté míry činí jednoduší a lépe distribuovatelnou.

Hlavní funkcí této aplikace je zrychlit a do jisté míry zjednodušit distribuci evolučních algoritmů, kdy s využitím většího množství procesorů dojde ke zvýšení výpočetního výkonu jako celku.

Celá aplikace je díky použitým technologiím do budoucna škálovatelná a díky neustálému inovování vývojového prostředí Microsoft Visual Studio .NET i značně rozšiřitelná.

Aplikaci je možné v budoucnu rozšířit například o celkové statistiky počtu připojených klientů, počtu zaslaných požadavků, průměrné doby vyřízení požadavku a mnoho dalšího.

Věřím, že tato aplikace najde v dnešní moderní době svoje uplatnění a bude sloužit nejen k distribuci evolučních algoritmů, ale také k distribuci jiných zadání procesorově náročných úloh.

ZÁVĚR V ANGLIČTINĚ

The main purpose of this thesis was about creating distributed computing system between central application and set of client applications. Whole solution consists of two parts. The first part is central application which sends and monitors set of client applications (workstations). Client application is listening on the specified port and is waiting for any commands from central application. Both parts are written using Microsoft .NET platform using Windows Communication Foundation and Windows Forms (only central application) technologies. None of both parts (client or central applications) do not use any database servers, which make it in some way more distributable.

One of the main goals of this application is in making distribution of evolutionary algorithm easier in some way and faster when we use power of distributed computing.

Whole application is expandable using Microsoft .NET platform and Visual Studio .NET innovation.

This software project (central and client part) may be expanded in future. Some new features like global statistics of average response time, number of requests and much more.

I believe that this application will find it’s place in this modern time and will serve not just only for distribution of evolutionary algorithms but also any processor extensive tasks.

SEZNAM POUŽITÉ LITERATURY

[1] Wikipedie [online]. 12.5.2010 [cit. 2010-06-07]. Distribuovaný výpočet. Dostupné z WWW: <http://cs.wikipedia.org/wiki/Distribuovan%C3%BD_v%C3%BD po%C4%8Det>.

[2] BOINCstats [online]. 2007 [cit. 2010-06-07]. Kombinované BOINC - přehled kreditu. Dostupné z WWW: <http://cz.boincstats.com/stats/project_graph.php?pr=bo>.

[3] PC Tuning [online]. 26.8.2008 [cit. 2010-06-07]. BOINC - počítače všech zemí, spojte se. Dostupné z WWW: <http://pctuning.tyden.cz/ilustrace3/forest/boinc/prijimac2.jpg>.

[4] PC Tuning [online]. 26.8.2008 [cit. 2010-06-07]. BOINC - počítače všech zemí, spojte se. Dostupné z WWW: <http://pctuning.tyden.cz/index.php?option=com_content&task=view&id=11449 &Itemid=96>.

[5] Enterprise Web 2.0 [online]. 2009 [cit. 2010-06-07]. Cloud Computing and future. Dostupné z WWW: <http://blogs.zdnet.com/Hinchcliffe>.

[6] Wikipedia [online]. 2009 [cit. 2010-06-07]. Cloud Computing. Dostupné z WWW: <http://en.wikipedia.org/wiki/Cloud_computing>.

[7] Jiko Blog : software as a service (SaaS), web 2.0, enterprise 2.0 [online]. 7.10.2009 [cit. 2010-06-07]. Cloud computing čtyřikrát jinak. Dostupné z WWW: <http://www.abako.cz/blog/767/cloud-computing-ctyrikrat-jinak/>.

[8] ŠVEC, Petr. ITBiz magazín [online]. 5. březen 2010 [cit. 2010-06-07]. Jak se dělá cloud computing. Dostupné z WWW: <http://www.itbiz.cz/par-slov-ke-cloud-computingu>.

[9] ROBINSON, Simon, et al. C# : Programujeme profesionálně. První vydání. Brno : Computer Press, a.s., 2003. 352 s. ISBN 80-251-0085-5.

[10] ESPOSITO, Dino. ASP.NET a ADO.NET : tvorba dynamických webových stránek. První vydání. Praha : Grada Pupblishing a.s., 2003. 352 s. ISBN 80-247-0474-9.

[11] Journal of Object Technology [online]. 2006 [cit. 2010-06-07]. .NET Remoting. Dostupné z WWW: <http://www.jot.fm/issues/issue_2006_04/article3/images/figure2.gif>.

[12] CHAPPELL, David. Microsoft MSDN [online]. 2010 [cit. 2010-06-07]. Introducing Windows Communication Foundation in .NET Framework 4. Dostupné z WWW: <http://msdn.microsoft.com/library/ee958158.aspx>.

[13] ZELINKA, Ivan. Umělá inteligence : v problémech globální optimalizace. 1. vydání. Praha : Ben, 2002. 192 s. ISBN 80-7300-069-5.

[14] Gymnázium Olomouc-Hejčín. Programování se zaměřením na .NET a jazyk C# [online]. 18. října 2006 [cit. 2010-06-07]. Windows Forms. Dostupné z WWW: <http://projektysipvz.gytool.cz/ProjektySIPVZ/Obrazky/Programovani/OvladaciPrvkyOdvozeniButtonBase.gif>.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

AIDS	Acquired Immune Deficiency Syndrome – syndrom získaného selhání imunity je nakažlivá a v současné době neléčitelná choroba postihující lidskou imunitu.
BOINC	Berkeley Open Infrastructure for Network Computing – infrastruktura pro distribuované výpočty.
BOINC	Berkeley Open Infrastructure for Network Computing je infrastruktura pro distribuované výpočty, která umožňuje provozovat projekty jako je SETI@home.
CaaS	Communication as a Service – komunikační služba nabízena v rámci Cloud řešení.
Cluster	Seskupení volně vázaných počítačů, které spolu úzce spolupracují, takže navenek mohou pracovat jako jeden počítač.
COM	Component Object Model – objektový model pro softwarové produkty na platfomě Microsoft Windows.
DCOM	Distributed Component Object Model – objektový model vyvinutý společností Microsoft distribuci COM komponent.
DLL	Dynamic-link library – sdílená knihovna používaná v prostředí operačních systémů Microsoft Windows.
DNA	Deoxyribonukleová kyselina – je nositelkou genetické informace všech organismů s výjimkou některých buněčných.
FLOPS	Floating Point Operations Per Second – zkratka vyjadřující počet operací s plovoucí čárkou za sekundu.
HTML	Hypertext Markup Language – jazyk popisující zdrojové stránky na internetu.
IBM	International Business Machines Corporation - je přední světová společnost v oboru informačních technologií.
LGPL	Lesser General Public License – licence svobodného software.

MSIL	Microsoft Intermediate Language – nejnižší člověkem čitelný programovací jazyk definovaný specifikací Common Language Infrastructure používaný projekty .NET Framework
MSMQ	Microsoft Message Queuing – je doručovací fronta vyvinutá společností Microsoft pro využití na Windows Serveru.
NASA	National Aeronautics and Space Administration - americká vládní agentura zodpovědná za americký kosmický program a všeobecný výzkum v oblasti letectví.
Peer-to-Peer	Způsob komunikace, kdy není využit centrální prvek. Každý klient komunikuje s každým.
SaaS	Software as a Service – poskytování softwarových služeb v Cloud Computingu jako jednotného řešení.
SETI@home	Search for Extraterrestrial Intelligence at home – hledání mimozemské inteligence u sebe doma.
SMTP	Simple Mail Transfer Protocol – je internetový protokol pro emailovou komunikaci mezi počítači. Standardně používá TCP port 25.
SOAP	Simple Object Access Protocol – standard pro přenos XML dat pomocí internetu.
SOMA	Samo-Organizující se Migrační Algoritmus – algoritmus, jehož činnost je založena na geometrických principech.
TCP	Transmission Control Protocol – základní protokol pro komunikaci na Internetu. Protokol zajišťuje spolehlivé doručení paketů ve správném pořadí.
TCP	Transmission Control Protocol – základní protokol pro komunikaci na Internetu. Protokol zajišťuje spolehlivé doručení paketů ve správném pořadí.
URL	Uniform Resource Locator – je řetězec znaků s definovanou strukturou, který slouží k přesné specifikaci umístění zdrojů informací
WCF	Windows Comunication Foundation – nová technologie od společnosti

	Microsoft pro platformu .NET, která umožňuje komunikaci mezi aplikacemi pomocí definovaných služeb.
XML	Extensible Markup Language – značkovací jazyk, který byl vyvinut a standardizován konsorciem W3C.

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: UKÁZKA KÓDU FORMULÁŘE FRMSETTINGS

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace PR.DP.Central
{
    public partial class frmSettings : Form
    {
        public frmSettings()
        {
            InitializeComponent();
        }

        private void frmSettings_Load(object sender, EventArgs e)
        {
            if
(string.IsNullOrEmpty(ConfigurationSettings.AppSettings["centralIpAddress
"]))
            {
                this.txtIpAddress.Text = "Není zadána";
            }
            else
            {
                this.txtIpAddress.Text =
ConfigurationSettings.AppSettings["centralIpAddress"];
            }
        }

        private void btnSave_Click(object sender, EventArgs e)
        {

            frmMain frm =
Application.OpenForms.OfType<frmMain>().SingleOrDefault();
            ListBox.ObjectCollection computers =
frm.GetClientComputers();

            if (computers.Count != 0)
            {
                CentralToClient ctc = new CentralToClient();
                for (int i=0;i<computers.Count;i++)
                {
                    ctc.UpdateClientIpAddress(this.txtIpAddress.Text,
computers[i].ToString());
                }
            }
            // Change value
            Configuration config =
ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);

            // Change value
            config.AppSettings.Settings["centralIpAddress"].Value =
this.txtIpAddress.Text;

            // Save changes
```

```
            config.Save(ConfigurationSaveMode.Modified);

            // Refresh Sections
            ConfigurationManager.RefreshSection("appSettings");

            MessageBox.Show("Případné změny byly uloženy!");
        }

        private void btnGetIP_Click(object sender, EventArgs e)
        {
            frmSettingsIpAddress frmIP = new frmSettingsIpAddress();
            frmIP.ShowDialog();
        }

        public void SetIpValue(string value)
        {
            this.txtIpAddress.Text = value;
        }
    }
}
```