MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Možnosti monitorování běžících procesů a reakce na zjištěný stav

DIPLOMOVÁ PRÁCE

Bc. Vladimír Lambert

Brno, podzim 2014

Prohlášení

Prohlašuji, že tato diplomová práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Vedoucí práce: Ing. Mgr. et Mgr. Zdeněk Říha, Ph.D.

Poděkování

Rád bych poděkoval společnosti SEACOMP s.r.o. za zajímavé zadání diplomové práce a za průběžně poskytovanou zpětnou vazbu. Také bych chtěl poděkovat vedoucímu této práce Ing. Mgr. et Mgr. Zdeňku Říhovi, Ph.D. za poskytnuté rady.

Shrnutí

Tato práce se zabývá možností monitorování běžících procesů. Po představení souvisejících technologií je popsán zvolený způsob řešení tohoto problému a jeho vybrané části jsou rozebrány detailněji.

Klíčová slova

WCF, C#, monitorování procesů, Windows service

Obsah

Kapitola 1

Úvod

Tato diplomová práce byla vytvořena v rámci průmyslové spolupráce se společností SEACOMP s.r.o., která se zabývá vývojem informačních systémů pro nemocniční prostředí i k jiným účelům.

Požadavkem zadavatele práce bylo, aby byl vytvořen program, který by monitoroval běh těchto systému. Pokud by zvolené parametry sledovaného procesu dosáhly hodnot, které byly předem stanoveny, měl tento program automaticky vykonat některou z možných akcí.

Komunikace s vytvořenou aplikací měla být možná i vzdáleně. Tím by bylo umožněno spouštět a zastavovat monitorování procesů a nastavovat jeho vlastnosti z programu, který by mohl být spuštěn na jiném počítači.

Bylo požadováno, aby tento obslužný program byl vybaven grafickým uživatelským rozhraním a aby všechny části byly vytvořeny za pomoci programovacího jazyka C#.

V průběhu doby, ve které byla tato práce tvořena, probíhaly pravidelné konzultace se zadavatelem, který na těchto zpřesňoval své požadavky na výsledný produkt a byl seznamován s jednotlivými dílčími řešeními.

Zadavatel si nepřál, aby byl zveřejněn zdrojový kód vytvořených aplikací, zejména služby, která provádí monitorování běžících procesů.

První kapitolou této diplomové práce je úvod. Neboť ten je vámi právě teď čten, jeho popis není potřebný.

V druhé kapitole jsou popsané technologie, které byly použité při implementaci této práce. Jedná se o jazyk C#, Windows Communication Foundation a Extensible Markup Language. Značná část této kapitoly je věnována WCF, což je technologie, která umožňuje snadnou komunikaci dvou procesů běžících na platformě .NET. Vzhledem k tomu, že cílem této práce bylo vytvořit dvě navzájem propojené aplikace, je tato velmi důležitou součástí konečného řešení.

V kapitole třetí je představeno několik programů, které by mohly poskytovat řešení na zadaný problém. Jedná se pouze o malý výběr, neboť funkci monitorování stavu spuštěných procesů jich nabízí nepřeberná řada. U každého z uvedených programů jsou vypsány důvody, pro které nemohl být tento použit. Ostatní neuvedená řešení nebyla využita z důvodů shodných s těmi uvedenými v této části.

Čtvrtá kapitola popisuje samotnou tvorbu požadovaných programů. Jsou zde do větší hloubky rozebrány požadavky zadavatele práce a jsou popsány jednotlivé aspekty

konečného řešení, především zvolený způsob komunikace a vzhled a nabízené funkce vytvořeného grafického uživatelského rozhraní

Pátou a poslední kapitolou této práce je pak závěr. V tom je zhodnocen výsledný produkt a popsány možné cesty jeho dalšího vývoje.

Kapitola 2

Použité technologie

V této kapitole jsou popsány technologie použité při implementaci řešení. Jedná se o programovací jazyk C#, jehož použití bylo požadováno zadavatelem práce. Windows Communication Foundation, které je použito pro zprostředkování komunikace mezi dvěma programy, které byly vytvořeny. A XML jako formát, ve kterém jsou ukládána nastavení serverové části a také data, která jsou produkována.

2.1 C#

C# je objektově orientovaný, typově bezpečný jazyk, který lze použít k rozličným účelům. [1] Syntax jazyka je velmi podobná jazykům C++ nebo Java a stejně jako v ostatních objektově orientovaných jazycích se i v tomto uplatní principy zapouzdření dědičnosti a polymorfismu.

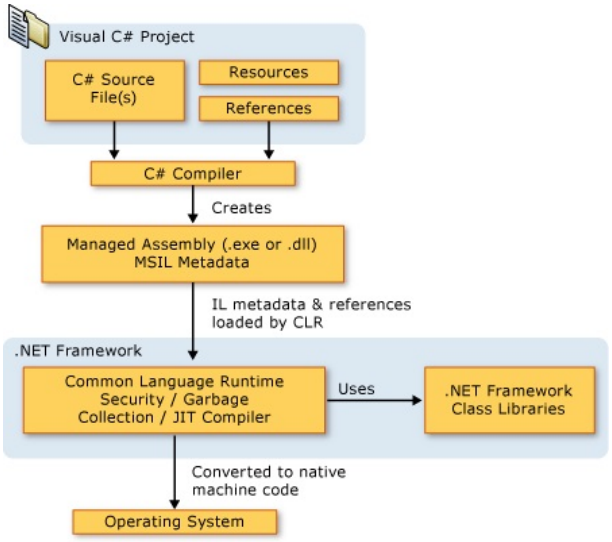
Programy vytvořené v jazyce C# běží ve frameworku .NET, který je součástí operačního systému Windows a obsahuje virtuální běhové prostředí se jménem *Common Language Runtime (CLR)*, které provádí *just in time* kompilaci a překlad do strojových instrukcí, a sadu knihoven. Tento framework podporuje i jiné programovací jazyky, jako příklad je možné zmínit C++ nebo F#. Bez ohledu na to, v kterém z nich je program napsán, je třeba nejprve uskutečnit překlad do takzvaného mezijazyka *IL (Intermediate Language)*. Takto vzniklý soubor, který může být označen příponou .exe je pak při spouštění nahrán do *CLR*, který se postará o jeho jeho provádění. [2] Díky tomuto přístupu lze používat knihovny napsané v jiném jazyce, než je kód programu, pokud je výsledek v obou případech přeložen do *IL*.

2.2 WCF [3] [4] [5]

WCF, neboli Windows Communication Foundation byla představena společností Microsoft v roce 2003. Hlavními cíli, kterých mělo tímto být dosaženo jsou:

Sjednocení existujících technologií

WCF řeší problém roztříštěnosti distribuovaných řešení tím, že představuje jednotný rámec. Při změně způsobu komunikace, například z *HTTP* na *TCP*, pak není třeba znovu



Obrázek 2.1: Překlad a spouštění programu napsaného v C# [2]

psát celé sekce kódu, ale postačí pouze změnit několik parametrů v nastavení komunikačního kanálu. Také odpadá potřeba pro to, aby se vývojáři distribuovaných systému museli seznamovat s novým API vždy, když chtějí změnit způsob, kterým spolu jednotlivé komponenty komunikují.

Interoperabilita napříč platformami

Většina velkých softwarových společností vyvíjí programy za použití proprietárních komunikačních protokolů, které jsou úzce svázány s určitou platformou. To může představovat problém, neboť často není možná komunikace s programy, které běží na platformách jiných. V těchto společnostech pak často dochází k tomu, že ani jejich jednotlivé systémy nejsou schopny vzájemné komunikace, protože byly vytvořeny nebo zakoupeny v různých časových obdobích. Podnikové systémy se pak skládají z mnoha navzájem nepropojených a nekomunikujících částí.

Schopnost vzájemné komunikace jednotlivých systému se však v poslední době stává stále více důležitou. Je tak nezbytné, aby nově vytvořené aplikace byly schopné spolupracovat s těmi již existujícími a to bez ohledu na to, v jakém programovacím jazyku jsou napsány. V reakci na tyto potřeby pak velké softwarové společnosti založily organizaci *Web Services Interoperability (WS-I)*, která vytvořila řadu standardů, které měly umožnit komunikaci aplikací běžících na různých platformách. Tyto standardy jsou specifické svou jednoduchostí, malou velikostí a modularitou, která umožňuje implementovat pouze ty části, které budou výsledným produktem používány. Služby, které aplikace nabízí jsou popsány pomocí *WSDL (Web Services Description Language)*. Při navázání spojení si tak může druhá stana specifickým dotazem vyžádat seznam metod, které rozhraní aplikace poskytuje. Tento seznam je ve formátu XML. Vzhledem k již výše popsané snaze o sjednocení a interoperabilitu Windows Communication Foun-

dation tyto standardy podporuje.

Možnost vývoje SOA

Zkratka SOA zastupuje pojem Service Oriented Architecture. Jedná se o paradigma, které stanovuje způsob, jakým se vyvíjejí rozsáhlé informační systémy nebo jakékoli větší aplikace.

Podle tohoto přístupu není praktické vytvářet monolitické systémy, neboť jejich tvorba zpravidla vyžaduje několik let a ve výsledku plní pouze úzce specifikované úkoly. V poslední době však vyvstává potřeba rychlé adaptace funkcí, které systém nabízí a SOA je princip vývoje softwaru, který se snaží tento požadavek řešit.

Pokud systém implementuje principy SOA, je tento složen z jednotlivých služeb. Službou se v tomto případě rozumí autonomní systém, který na vstupu přijímá jeden nebo více požadavku a vrací jednu nebo více odpovědí za použití předem definovaných rozhraní. Každá z těchto služeb tak tedy může být změněna nezávisle na službách ostatních za účelem toho, aby byla schopna plnit nové požadavky. Samozřejmě za předpokladu, že bude implementovat stále stejné (případně širší) rozhraní.

Dosud popsané principy mohou vzbuzovat dojem, že SOA je v podstatě popisem stejné architektury jako systém složený z komponent (paradigma *component-based development*). Hlavním rozdílem SOA oproti tomuto přístupu jsou již popsané zprávy, které nejsou platformově závislé a jejichž syntaxe je definována zmíněnými standardy. To vede k možnosti vystavět systém, který není pevně spojen a funguje napříč jednotlivými technologiemi a platformamy.

Následující zásady SOA byly navrženy stejným týmem, který pracoval na vývoji WCF, které tak samozřejmě umožňuje tvorbu systémů tyto naplňující.

- **Hranice služby jsou explicitní.** Veškeré interakce se službou jsou prováděny prostřednictvím definovaného rozhraní, které by mělo být relativně malé a snadné na použití. Komunikace by nemělo být dosaženo za pomoci vzdáleného volání procedur, ale výměnou zpráv.

- **Služba je autonomní.** Jednotlivé služby jsou na sobě nezávislé, je tak možné změnit pouze určitou službu a při zachování definovaného rozhraní by měl nový systém stále fungovat. Již zveřejněné metody rozhraní by neměly být měněny.

- **Služby sdílejí kontrakt, ne implementaci.** Služby spolu nekomunikují za použití sdílených tříd, ale pomocí zpráv s předem stanovenou syntaxí. Formát těchto zpráv je definovaný pomocí WDSL. Slovem kontrakt se pak označuje dokument, který popisuje nejen potřebný formát zpráv, ale také způsob, jakým si mezi sebou jednotlivé služby tyto zprávy vyměňují. Tento kontrakt by pak měl zůstat neměnný.

- **Kompatibilita služeb je založena politikou.** V určitých případech není možné popsat požadovanou interakci služeb pouze pomocí WSDL. V takovém případě je pak možno přistoupit k použití politik, které umožňují různé chování služby podle toho, s kým tato právě komunikuje. Je tak možné například povolit volání určitých metod jen specifickým službám.

Windows Communication Foundation je pak prvním programovacím nástrojem, který byl od základu vytvořen s cílem umožnit vývoj systémů řídících se pravidly SOA. I když koncept Service Oriented Architecture je znám již několik let, je v současnosti stále považován za nejlepší možný přístup k tvorbě rozsáhlých a distribuovaných systémů.

Služba WCF

Program, který implementuje WCF se v používané terminologii nazývá službou. Tato služba poskytuje metody z definovaného rozhraní a splňuje zásady SOA, které jsou popsány výše. Tato služba pak pracuje tak, že je neustále připravena přijímat požadavky od klientstké aplikace, která by chtěla využít některou z jejích nabízených funkcionalit. To samozřejmě nebrání tomu, aby služba sama v čase, kdy zrovna nepřijímá žádné požadavky, nevykonávala svou vlastní činnost.

Windows Communication Foundation služby komunikují se svým okolím vždy pomocí zpráv nezávisle na tom, zda se nachází na stejném počítači, nebo jsou rozprostřeny na síti.

Endpoint

Každá WCF služba, která chce mít možnost komunikovat se svým okolím musí mít jeden nebo více endpointů. Veškerá komunikace s WCF službou probíhá právě prostřednictvím těchto endpointů, které služba má. Endpointy poskytují klientům přístup k funkcionalitám, které WCF služba nabízí. [6] Každý endpoint se skládá z následujících částí:

- **Adresa**, která indikuje, kde může být endpoint nalezen a kam jsou posílány zprávy jemu určené.
- **Binding**, který specifikuje způsob, jakým spolu služba a klient komunikují
- **Kontrakt**, který popisuje metody, které služba nabízí a formát přenášených dat.
- **Chování**, které specifikuje, jaké jsou detaily implementace endpointu.

Jednotlivé vlastnosti endpointu je pak možné nastavit přímo v kódu programu implementujího WCF službu, nebo je lze definovat v XML souboru, který je součástí vytvářeného řešení.

Adresa

Vzhledem k tomu, že služba WCF poskytuje své metody zásadně pomocí zasílaných zpráv, je nutné, aby tato měla svou adresu, na které může tyto zprávy přijímat. Přesněji, adresu má konkrétní endpoint zmiňované služby. Tato adresa musí být unikátní.

Adresa endpointu je specifikována pomocí URI a její přesná podoba závisí na protokolu, který je použitý pro přenos zpráv (HTTP, TCP, MSMQ[1]).

Binding

Pomocí binding jsou stanovovány parametry komunikace. Každý endpoint WCF služby musí mít binding specifikovaný. Tvoří ho následující prametry:

- **Kódování**, které stanovuje, jakým způsobem jsou zasílané zprávy kodované (například zda textově nebo binárně).
- **Transportní protokol**, který určuje, který z transportních protokolů bude použit k přenosu zpráv (například TCP či HTTP).
- **Další protokoly**, které můžou být pomocí binding definovány za účelem zvýšení bezpečnosti přenášených zpráv.

Kontrakt

Kontrakt popisuje rozhraní služby, které endpoint nabízí. Windows Communication Foundation umožňuje definovat čtyři základní druhy kontraktů.

Service contract

Tento kontrakt popisuje operace, které služba přes příslušný endpoint nabízí. V jazyce C# je toto implementováno standardní třídou rozhraní (interface), do které jsou přidána klíčová slova, která definují, že toto konkrétní rozhraní bude poskytováno WCF službou. Na následujícím příkladu je zobrazena část kontraktu služby, která byla vytvořena v rámci této práce.

1. Microsoft Message Queuing

```
[ServiceContract]
public interface IWatchDog
{
    [OperationContract]
    ProcessStats GetProcessStats(int pid);

    [OperationContract]
    bool StartMonitoring(string processName);
}
```

Data contract

Tímto kontraktem je možné specifikovat vlastní datové typy, které služba používá při komunikaci. Jednoduché datové typy jazyka C# je možné používat bez jejich specifikace pomocí tohoto kontraktu. Pokud však má služba vracet jako hodnotu nějaký nově definovaný typ, případně takovýto typ přijímat jako vstupní parametr některé z metod, které tato poskytuje, je třeba takovýto typ označit pomocí kontraktu. Jak je vidět na následujícím příkladu, opět se jedná o standardní třídu jazyka C# rozšířenou o potřebná klíčová slova (Ani v tomto případě se nejedná o celou třídu, ale pouze o její fragment, který je použit k názorné demonstraci požadované syntaxe).

```
[DataContract]
public class ProcessStats
{
    [DataMember]
    public int Pid { get; set; }
    [DataMember]
    public String Name { get; set; }
    [DataMember]
    public int HandleCount { get; set; }
}
```

Message contract

Message contract umožňuje definovat strukturu odesílaných a přijímaných zpráv. Tento kontrakt není třeba definovat, neboť WCF dokáže podle zadaných dat zprávu vytvořit bez nutnosti zásahu programátora. Pomocí tohoto kontraktu je možné naprosto kontrolovat jaká bude struktura SOAP[2] zprávy. V programu, který byl vytvořen jako součást této práce, není tento kontrakt používán.

2. Simple Object Access Protocol

Fault contract

Pomocí tohoto kontraktu je možné definovat jak budou vypadat chybové zprávy, které může služba zasílat klientovi. Pokud by chtěla některá z metod, které služba poskytuje, ve zvláštních případech oznámit klientovi, že došlo k chybě, je třeba, aby tento byl schopen chybovou zprávu příjmout i v případě, že struktura této zprávy je odlišná od struktury zprávy, kterou klient očekává jako odpověď na svůj požadavek. K tomuto účelu slouží právě fault contract, který umožňuje označit metodu, která může chybovou zprávu vracet. Použití je vidět na následujícím příkladu. [7]

```
[ServiceContract]
public interface ICalculator
{
    [OperationContract]
    int Add(int n1, int n2);
    [OperationContract]
    [FaultContract(typeof(MathFault))]
    int Divide(int n1, int n2);
}
```

Typ zprávy, která je předána jako chybové hlášení je pak nutné opět označit pomocí Data contract, pokud není předávaná zpráva typu definovaného jazykem C#. V programu, který byl vytvořen v rámci této práce není tento typ kontraktu použit.

Chování

Pomocí specifikace chování (behavior) je možné určit, jak se WCF služba zachová v určitých situacích.

Lze tak například nastavit, zda má služba mít možnost vytvářet více svých instancí, nebo budou veškeré požadavky řešeny pouze v jednom vlákně.

Je možné specifikovat zda má klient celou dobu komunikovat s tou samou instancí služby, nebo zda má být pro každý jeho dotaz vytvořena nová.

V případě, že je možné, aby služba měla více instancí, je možné určit jejich maximální počet.

Hostování WCF služby

Služba Windows Communication Foundation sama o sobě není schopná vykonávat žádnou z činností, které byly doposud popsány. Především nemůže odpovídat na zaslané zprávy. K tomu, aby byla služba schopná komunikovat, je třeba, aby tato byla hostována v nějaké běžící aplikaci. Základních možností, jak hostovat WCF službu je několik:

- **Self-hosting** Toto je nejjednodušší možnost, jak hostovat WCF službu. Jde o případ, kdy je služba hostována ve své vlastní aplikaci (například konzolové nebo

vytvořené za použití Windows Forms). Z toho vyplývá relativní jednoduchost zapínání a zastavování této služby prací s touto aplikací. Další velmi nezanedbatelnou výhodou je možnost snadného debuggovaní této WCF služby.

Na druhou stranu k tomu, aby byla takto hostovaná služba dostupná je třeba, aby byla hostující aplikace neustále spuštěná.

- **Hostování ve Windows service** Hned pro začátek je třeba si uvědomit, že Windows service a WCF service jsou dva různé pojmy, a proto nemohou být zaměňovány. Windows service je proces obhospodařovaný operačním systémem. Windows používají tyto služby na podporu nejrůznějších funkcí operačního systému jako je komunikace po síti, práce s USB a podobně.

  Vytvoření Windows service, která by měla hostovat WCF službu není nikterak komplikované a oproti self-hostingu má tento způsob řešení několik nezanedbatelných výhod.

  Zejména je velmi snadné nastavit Windows service tak, aby tato byla automaticky spuštěna při každém startu systému.

  Service Control Manager, což se je součást systému Windows, která dohlíží na aktivní Windows služby a povoluje jejich instalaci, může Windows service, která hostuje WCF službu, v případě jejího pádu automaticky restartovat. Také umožňuje nastavení oprávnění, která bude běžící služba mít.

  Oproti tomu je v případě, kdy bylo rozhodnuto, že k hostování bude použita Windows service, složitější službu na počítači zprovoznit. Na rozdíl od sefl-hostingu, kdy stačí výslednou aplikace pouze přeložit a následně spustit, Windows služby je třeba do systému instalovat.

  Tento způsob hostování byl zvolen v této diplomové práci, protože výsledek, kterého je takto dosaženo, nejvíce odpovídá parametrům zadání a tento způsob řešení byl dokonce zadavatelem práce doporučen.

- **Hostování za použití IIS** IIS neboli Internet Information Services je webový server vytvořený společností Microsoft. Všeobecně je hostování WCF služeb tímto způsobem považováno za nejvíce převažující.

  Tento způsob hostování je nejčastěji volen, pokud má WCF služba poskytovat přístup ke svému rozhraní prostřednictvím počítačové sítě. V takovém případě však lze, jak bylo zmíněno, použít i jakoukoli ze dvou dříve popsaných metod. Tato metoda je však pro tuto eventualitu nejlépe vybavena.

- **Hostování ve Windows Azure** Microsoft Windows Azure je cloudová platforma, která zprostředkovává hostování a spouštění aplikací. K vytváření aplikací, které by mohly být spuštěny na Windows Azure, což je operační systém, který tato platforma používá, je třeba speciální vývojové prostředí.

2.3 XML [8]

XML je označení pro Extensible Markup Language, jedná se o jednoduchý značkovací jazyk odvozený z SMGL (Standard Generalized Markup Language [9]), konsorciem W3C. [10]

Mezi hlavní způsoby použití jazyka XML patří reprezentace dat, například v konfiguračních souborech (toto použití bylo aplikováno při řešení praktické části této práce), a přidávání metadat do dokumentů, toto užití umožňuje například volbu stylu textu.

Jazyk XML byl vytvořen, aby řešil problém vzájemné komunikace tím, že umožní přenos dat v jednotném formátu namísto používání různých řešení v různých programovacích jazycích. Hlavní předností je, že tento formát je běžně čitelný a soubory, které ho využívají nejsou kódovány v binární, ale v textové podobě. To znamá, že k jejich zpracování nejsou potřeba zvláštní aplikace.

Toto vede nejen k tomu, že soubory v tomto formátu mohou být jednoduše čteny a upravováný lidmi, ale také k jejich jednoduššímu zpracování programy, které se rozhodnou použití tohoto jazyka upřednostnit před používáním souborů v binární podobě.

Hlavní nevýhodou tohoto přístupu je pak velikost vzniklých souborů, protože snadné čitelnosti je dosaženo přidávaním tagů do textových souborů a ukládání veškerých dat v textové podobě. To samozřejmě vede k tomu, že v porovnání s binárním souborem, který obsahuje stejná data, má soubor ve formátu XML znatelně větší velikost.

Struktura

K tomu, aby byl XML dokument považován za správně strukturovaný (well-formed) je třeba, aby tento splňoval kritéria stanovená konsorciem W3C. [11] Mezi nejdůležitější pravidla, bez jejichž dodržení nebude výsledný soubor považován za správně formátovaný, a nemusí tak být rozpoznán všemi aplikacemi, které s XML pracují patří například:

- Mohou být požity pouze znaky ze sady UNICODE a to v kódování UTF-8 nebo UTF-16
- V souboru může být pouze jeden kořenový element. Veškerý další obsah dokumentu musí být obsažen v tomto elementu. To může vést k problémům, zejména pokud je XML soubor použit k logování záznamů, jak tomu je například v praktické části této práce.
- Řidící znaky jazyka XML se v dokumentu nesmí vyskytovat na nepovolených místech, například uprostřed textu.
- Elementy, ze kterých se dokument skládá, musí byt uspořádány ve stromové struktuře. To znamená, že každý element, kromě kořenového, má právě jeden element za svého předka. Počet následníků, které element může mít není nikterak omezen. Jednotlivé elementy se také nesmějí navzájem překrývat. Tedy tag označující

konec určitého elementu se nesmí v souboru nacházet před tagem, který ukončuje element, který je jeho následníkem. Tedy jehož počáteční tag je umístěn mezi počátečním a ukončujícím tagem svého rodiče.

- Jména jsou citlivá na velikost písmen (case-sensitive) a nesmí v nich být obsaženy prázdné znaky

Mimo nutných podmínek existují ještě mnohá doporučení na strukturu dokumentu. Patří mezi ně například uvádění verze XML na začátku dokumentu.

Kapitola 3

Stávající řešení

V této kapitole budou popsány různé, již existující, technologie, jejichž použitím je možné dosáhnout zadaných cílů. Budou uvedeny důvody, které vedly k tomu, že tyto nakonec nebyly použity.

Zadavatel diplomové práce stanovil jako zásadní podmínku, že vytvořený program musí být spustitelný na operačním systému Windows a naprogramovaný za pomocí programovacího jazyka C#. V tomto jsou prý vytvořeny i ostatní systémy které mají být monitorovány, i když o pravdivosti tohoto tvrzení se autor práce dosud nemohl přesvědčit.

Podle požadavků zadavatele, měl být vytvořen program, který by umožňoval monitorování parametrů běžícího .NET procesu. V tomto konkrétním případě mělo být především možné monitorovat stav běžících služeb (Windows service). Tento program by následně naměřené hodnoty poté ukládal do specifického souboru v případě, že se by monitorovaný proces dostal do zvoleného stavu.

Kromě monitorování hodnot ve specifikovaných situacích bylo také požadováno, aby bylo možno jejich hodnotami podmínit vykonávání specifických akcí. Zejména restartování procesu, který dosáhl stanovených hodnot, které jsou považovány za nebezpečné.

Dále bylo požadováno, aby program, který bude vykonávat monitorování bylo možné spustit jako *Windows service*, což usnadňuje jeho automatické spuštění v okamžiku startu systému a na rozdíl od aplikací konzolových, Windows Forms či Windows Presentation Foundation při spuštění programu touto formou není automaticky vytvářeno žádné okno.

Parametry, které měly být monitorovány jsou:

- Počet spuštěných vláken
- Počet alokovaných popisovačů (handles)
- Velikost alokované paměti
- Vytížení CPU sledovaným procesem

Dalším požadavkem bylo, aby vytvořený program umožňoval přístup k zásobníku monitorovaného .NET procesu. Přesněji, aby bylo možné, v případě, že by monitorované parametry sledovaného procesu dosáhly stanovených hodnot, obsah tohoto zá-

sobníku vypsat do souboru. Bylo požadováno vypsání metod, které se na zásobníku nacházejí, a to pro jednotlivá *managed* vlákna monitorovaného procesu.

Existuje nepřeberné množství programů, které umožňují monitorování běžících procesů. V této kapitole tak samozřejmě není uveden jejich plný výčet. Tyto programy jsou si většinou vesměs podobné a to jak nabízenými funkcemi, tak grafickým zpracováním uživatelského rozhraní. Budou tedy uvedeny jen vybrané příklady. Veškerá uvedená řešení byla vyzkoušena spolu s několika dalšími, která nejsou zmíněna z důvodu, že jsou vesměs podobná těm uvedeným a neřeší problémy zmíněné u jednotlivých uvedených programů.

3.1 Správce úloh

Správce úloh, neboli Task Manager je velmi známý program, který je standardně přítomný na operačním systému Windows. Slouží k zobrazování běžících programů, procesů a služeb. Lze ho použít k monitorování výkonu počítače jako celku nebo ke sledování zvoleného programu nebo procesu. [12]

Z nutných parametrů umožňuje správce úloh monitorovat vše potřebné a je možné ho použít k vytvoření takzvaného souboru výpisu (dump), ve kterém je uložen stav zásobníku.

Tento program nebyl použit, neboť získání hodnot, které jsou zobrazovány je netriviální, navíc se nejedná o řešení, které by bylo možné upravovat. Soubor výpisu je také ve speciálním proprietárním formátu a k jeho otevření a prozkoumání jeho obsahu je nutné Visual Studio, což je vývojové prostředí (IDE) od společnosti Microsoft.

3.2 Process Explorer

Process Explorer[1] je dalším programem, který nabízí monitorování všech požadovaných parametrů a umožňuje i přístup k zásobníku.

Stejně jako v případě správce úloh se však jedná především o řešení, které má uživateli zprostředkovat grafickou reprezentaci stavu procesů, které běží na monitorovaném počítači a není vytvořen k tomu, aby bylo možné zobrazované informace využít jiným programem. Zdrojové kódy k tomuto programu navíc nejsou volně dostupné, a tak není možná ani jeho úprava za účelem plnění požadovaných funkcí.

3.3 Managed Stack Explorer

Managed Stack Explorer[2] slibuje možnost přístupu k zásobníkům jednotlivých vláken. Tento program však dlouhou dobu nebyl aktualizován, a tak na současných systémech

1. dostupný například na: http://technet.microsoft.com/cs-cz/sysinternals/bb896653.aspx
2. dostupný z: https://mse.codeplex.com/

nepracuje. Podle stránek produktu je tento cílen na .NET framework verze 2.0 přičemž současná verze je 4.5.

Je možné najít i pokusy o aktualizaci tohoto nástroje. Jako příklad lze uvést CLR Stack Explorer [13]. Tento nástroj však není dokončen, nejsou k dispozici jeho zdrojové kódy a jeho běh je velmi pomalý.

Kapitola 4

Implementace

V této kapitole bude popsáno, jak probíhala implementace praktické části této práce. Z důvodu požadavku zadavatele nesmí být výsledné programy zveřejněny, s výjimkou částí, které neobsahují žádnou důležitou funkcionalitu.

4.1 Požadavky

Zadavatel požadoval, aby byl vytvořen program, který by mohl být spuštěn na počítači, na kterém běží procesy, které si přeje monitorovat. K tomuto programu mělo být možné se připojit z jiného počítače a vzdáleně měnit parametry monitorování, případně spustit monitorování nové či ukončit právě probíhající.

Vytvořený program měl periodicky ve stanovených časových intervalech kontrolovat požadované parametry sledovaného procesu a v případě, že by některý z těchto parametrů dosáhl zadaných hodnot, měla být provedena definovaná akce.

Touto akcí může být uložení hodnot definovaných parametrů, restartování monitorovaného procesu nebo spuštění připraveného skriptu. Tato varianta byla do zadání doplněna později.

Také měl být vytvořen program, který se k monitoru procesů připojí. Tento program měl poskytovat grafické uživatelské rozhraní, které by zobrazovalo procesy běžící na počítači, na kterém je monitor spuštěn, za účelem možnosti výběru procesu, který má být monitorován. Pomocí tohoto programu, také mělo být možno nastavit, jaké parametry zvoleného procesu budou ukládány a při splnění jakých podmínek.

4.2 Postup

Vývoj probíhal ve spolupráci se zadavatelem a to tak, že nejprve byla vytvořena konzolová aplikace, která umožňovala, po zadání příkazu, získat určité parametry zvoleného procesu.

Poté, co bylo ukázáno, že požadavky, které zadavatel měl je možné splnit a že je možné získat přístup k hodnotám všech parametrů, na jejichž monitorování měl zadavatel zájem, bylo přikročeno k postupnému vývoji finální verze programu.

Byla vytvořena aplikace využívající Windows form, která sloužila jako klient a s pomocí které může uživatel provádět potřebné akce. Tato aplikace také zobrazuje seznam

běžících procesů a hodnoty sledovaných parametrů pro tyto procesy.

Také byla vytvořena Windows service, která měla sloužit k hostování WCF služby.

V neposlední řadě pak byla vytvořena samotná WCF služba, která implementuje veškerou potřebnou funkcionalitu.

Požadavky zadavatele se v průběhu práce mírně rozšiřovaly, a díky tomu byla funkcionalita programů postupně upravována, aby vyhovovala jeho novým přáním.

4.3 Zvolené technologie

Technologie zvolené pro implementaci praktické části této práce byly již vesměs popsány v předchozích kapitolách, zde bude jen uveden důvod, proč byly vybrány právě tyto.

C#

Tento konkrétní programovací jazyk byl vybrán proto, že zadavatel práce používá operační systém Windows a jeho současné systémy, které mají být monitorovány běží na platformě .NET. Použití C# tak mělo umožnit snadné začlenění výsledného produktu do používaných systémů.

WCF

Protože z požadavků vyplývalo, že v rámci této práce by měly být vytvořeny dva spolu navzájem komunikující programy, bylo nutné zvolit způsob, jakým se tyto budou dorozumívat. V úvahu zde připadala celá řada řešení. První verze programu, která byla implementována využívala ke komunikaci TCP socket.

Toto řešení však zadavatelem nebylo schváleno, neboť jeho součástí bylo zasílání zpráv ve vlastním textovém formátu. Bylo požadováno, aby zde byla možnost pro jiné programy vstoupit do této komunikace, což by v případě tohoto řešení vyžadovalo neuměrné nároky na vývojáře.

Nakonec bylo rozhodnuto, že bude využito služeb Windows Communication Foundation, protože ze všech nabízených možností vypadala tato varianta jako nejlepší a to jak z hlediska univerzálnosti použití vysledného programu, tak z hlediska náročnosti implementace.

Windows Forms

Zadavatel požadoval nejen funkční program, který by monitoroval zvolené procesy, ale také snadnou možnost tento program vzdáleně ovládat. Z tohoto důvodu bylo vytvořeno grafické uživatelské rozhraní, které zprostředkovává informace o stavu monitorovaných procesů a umožňuje upravovat parametry monitorování.

K vytvoření grafického rozhraní v programovacím jazyce C# připadají v úvahu dva nejrozšířenější způsoby. Jedním z nich je využití Windows Forms a tím druhým je po-

užití Windows Presentation Foundation. I když je Windows Presentation Foundation modernější a má širší použitelnost, bylo rozhodnuto o tom, že ve výsledném programu budou použity Windows Forms. K tomuto rozhodnutí vedlo to, že funkcionalita grafického uživatelského rozhraní nepatřila mezi hlavní cíle práce a také to, že autor má určité zkušenosti s Windows Forms a domníval se, že toto řešení bude pro daný účel postačovat.

MdbgEngine[1] [2]

Tato technologie nebyla popsána v kapitole *Použité technologie* neboť se o ní nepodařilo dohledat žádné zdroje, které by popisovaly jak tato vznikla a jak teoreticky funguje.

Tato knihovna byla použita za účelem přístupu k zásobníku monitorovaných procesů, což byl jeden z požadavků na funkcionalitu výsledného programu. Veškeré ostatní parametry je možné získat za pomoci třídy, která je v programovacím jazyce C# definována standardně, a to sice třídy *Process*. Tato umožňuje přístup k mnoha parametrům všech procesů, které jsou v operačním systému Windows spuštěny. Nicméně není možné tuto třídu použít k přístupu na zásobník zvoleného procesu.

Bylo zjištěno, že s pomocí této knihovny lze však přistupovat pouze k zásobníku procesů, které běží na platformě .NET. S tímto zjištěním byl zadavatel srozuměn a zkonstatoval, že to pro něho nepředstavuje problém, neboť veškeré procesy, které mají být monitorovány běží také na této platformě.

S jejím použitím je možné blíže přistupovat k jednotlivým vláknům zkoumaného procesu. Třida *Process*, která je standardně definovaná v jazyce C# a která se používá, pokud pokud vyvstane potřeba zjistit nejrůznější podrobnosti o běžícím procesu, umožňuje určit pouze počet vláken, která jsou v rámci zkoumaného procesu spuštěná. *MdbgEngine* naproti tomu umožňuje přístup k jednotlivým rámcům, které se nachází v zásobnících vláken monitorovaných procesů.

Lze tak zjistit jméno metody, jejímž zavoláním vnikly rámce na zásobníku a také typy proměnných (nebo konstant), se kterými byla metoda volána jako s parametry. Po podrobnějším zkoumání je možné získat i hodnoty těchto proměnných.

4.4 Komunikace

Již bylo napsáno, že ke zprostředkování komunikace mezi dvěma vytvořenými programy bylo použito služeb Windows Communication Foundation. Bylo však nutné definovat metody, které budou ke komunikace použité. Byly zváženy akce, které má být možno vzdáleně vykonat a navrženy následující metody poskytovaného rozhraní:

- `ProcessesStats GetProcesses();`

1. Dostupné na: https://www.nuget.org/packages/Microsoft.Samples.Debugging.MdbgEngine/
2. Zdrojové kódy knihovny lze najít na: http://www.symbolsource.org/Public/Metadata/NuGet/Project/Microsoft.Samples.Debugging.MdbgEngine/1.4.0.0/Release/Default/Microsoft.Samples.Debugging.MdbgEngine

Tato metoda umožňuje získání hodnot všech parametrů, které tento program umožňuje monitorovat pro všechny procesy, které jsou na počítači, na kterém WCF služba běží, spuštěny. Její použití je plánováno především s ohledem na to, že grafické uživatelské rozhraní má za úkol, zobrazovat seznam všech běžících procesů a tato metoda umožňuje jejich jednoduché získání.

- `ProcessStats GetProcessStats(int pid);`

  Pomocí této metody lze získat parametry jen pro jeden konkrétní proces. To o jaký proces se jedná je jednoznačně určeno zaslaným PID. Důvod, proč byla tato metoda implementována je ten, že grafické uživatelské rozhraní umožnuje také zobrazení podrobností o jednom konkrétním zvoleném procesu. Bylo by tak neefektivní v případě, že jsou požadovány informace pouze o jediném procesu zjišťovat a zasílat hodnoty pro procesy všechny.

- `bool SetMonitorSettings(MonitorSettings mSettings);`

  Metoda *SetMonitorSettings* umožňuje nastavení monitorování zvoleného procesu. Lze pomocí ní určit všechny požadované parametry monitorování pro proces s konkrétním jménem. Důvod, proč bylo v tomto případě jako identifikátor procesu zvoleno jméno namísto PID a o obsah objektu *MonitorSettings* bude popsán v dalším textu této práce.

- `MonitorSettings GetMonitorSettings(string processName);`

  Tato metoda umožňuje získat ze serveru informace o tom, jak je nastaveno monitorování procesu se zvoleným jménem. Je použita v případě, že si uživatel přeje změnit některé parametry monitorování k tomu, aby byl získán současný stav uložený na serveru.

- `bool StartMonitoring(string processName);`

  StartMonitoring, jak už název metody napovídá umožňuje vzdáleně spustit monitorování procesu se zvoleným jménem. Tuto akci je možné provést pouze v případě, že již existuje na serveru uložené nastavení, které by určovalo jakým způsobem má být proces monitorován.

- `bool StopMonitoring(string processName);`

  Pomocí této medody je pak možné ukončit monitorování procesu. Opět je třeba specifikovat jméno procesu, o který se jedná o tento také musí být ve chvíli, kdy se tato metoda volá aktivně monitorován, aby tato měla nějaký efekt.

- `List<string> GetStackTrace(int pid);`

Žádná z předchozích metod neposílala klientovi data o stavu zásobníku monitorovaného procesu. Vzhledem k tomu, že operace, kterou jsou tato data získávána je časově náročná a lze ji provádět pouze s procesy běžícími na platformě .NET, bylo rozhodnuto o tom, že tato funkcionalita bude vyčleněna do samostatné metody a tato data budou uživateli předložena jen na jeho přímou žádost. K identifikaci procesu je v tomto případě opět použito PID, neboť uživateli jsou předkládána data pouze o jednom procesu.

Výše uvedené metody používají ke komunikaci třídy, které byly vytvořeny speciálně pro tento účel jedná se o *MonitorSettings* a *ProcessStats*. Třída *ProcessStats* slouží k přenosu naměřených parametrů určitého procesu.

```
public class ProcessStats
{
    public int Pid { get; set; }
    public String Name { get; set; }
    public int HandleCount { get; set; }
    public int ThreadCount { get; set; }
    public int CpuUse { get; set; }
    public long WorkingSet { get; set; }
    public int MonitoringState { get; set; }
    public int Severity { get; set; }
}
```

Třída výše je pro zkrácení zapsána bez klíčových slov, které označují, že se jedná o *Data contract*. Toto označení umožňuje, aby byly objekty patřící do této třídy přenášeny v rámci komunikace zprostředkované WCF. Stejný postup byl zvolen i u tříd následujících.

Je vidět že *ProcessStats* obsahuje atributy, do kterých jsou ukládány naměřené hodnoty parametrů, které mohou být monitorovány. Kromě nich obsahuje ještě PID a jméno procesu kvůli snadnější identifikaci a tomu, aby uživatel byl schopen spustit monitorování procesu s konkrétním jménem. Jméno pro uživatele důležitým údajem, neboť podle něho se rozhoduje, který ze zobrazovaných procesů si vybere jako cíl akce, kterou se rozhodl provést.

Dalšími atributy jsou *MonitoringState*, který udává, zda je konkrétní proces v současnosti monitorován a zda je pro něj na serveru uložené nastavení, které by umožnilo případné spuštění monitorování.

Severity pak stanovuje, zda při monitorování procesu s konkrétním PID došlo v poslední době k výskytu nějaké události, jejíž sledování bylo nastaveno. Hodnota tohoto parametru pak udává závažnost té nejzávažnější události, která byla v rámcí tohoto procesu zachycena.

```
public class MonitorSettings
{
    public String Name { get; set; }
```

```
    public int ActivityState { get; set; }
    public List<MonitorEvent> Events { get; set; }
}
```

Třída *MonitorSettings* umožňuje, aby si klient se serverem vyměňovali informace i nastavení monitorování procesu s určitým jménem. Klient tak má možnost zjistit, jak konkrétní monitorování v určité době vypadá a pokud si to uživatel přeje, je možné toto nastavení změnit a zaslat ho zpět na server.

Důvodem, proč bylo vybráno jako identifikátor monitorovaného procesu jméno, je přání zadavatele. Hlavní myšlenkou za tímto rozhodnutím je, že monitorovaný proces může být ukončen a následně spuštěn. V takovémto případě je tento s největší pravděpodobností spuštěn s jiným PID, než s jakým běžel předtím. Aby tak monitorování pokračovalo i po této události, je třeba zvolit jiný identifikátor. Jméno bylo vybráno z toho důvodu, že toto zůstane i po opětovném spuštění procesu stejné.

Vedlejším efektem tohoto rozhodnutí je, že může být monitorováno více procesů se stejným jménem současně. K této situaci dochází u procesů, které jsou spuštěny ve více než jedné instanci nebo u těch se shodným jménem.

ActivityState stejně jako v případě předchozí třídy označuje stav monitorování zvoleného procesu. Tedy jestli je tento aktivně monitorován, případně, zda je pro něj na serveru uloženo nastavení pro případné monitorování. Důvodem, proč je tento atribut přítomný i v této třídě je, že program fungující jako server má uloženy *MonitorSettings* pro všechny procesy (podle jména), pro které uživatel nastavil parametry monitorování a na základě hodnoty této proměnné rozhoduje, zda mají být procesy s příslušným jménem monitorovány.

```
public class MonitorEvent
{
    public int Frequency { get; set; }
    public int Severity { get; set; }
    public List<string> Conditions { get; set; }
    public List<string> Actions { get; set; }
}
```

Kolekce složená s objektů této třídy je atributem třídy *MonitorSettings*. Důvodem je to, že pro každý proces může být definováno více událostí, které jsou sledovány. Každou takovou událost pak definuje instance třídy *MonitorEvent*. Pomocí té je možné stanovit frekvenci se kterou se má kontrolovat, zda událost nastala. Také lze zvolit závažnost, která je výskytu této události přisouzena.

Instance této třídy také obsahuje seznam podmínek, jejichž splnění se kontroluje v intervalech, které stanovuje frekvence a seznam akcí, které mají být provedeny v případě, že by byly všechny podmínky splněny.

Možnými akcemi, které současná verze programu podporuje jsou:

- Zalogování počtu spuštěných vláken
- Zalogování počtu alokovaných popisovačů

- Zalogování velikosti alokované paměti
- Zalogování vytížení CPU
- Zalogování výpisu zásobníku
- Restart procesu
- Spuštění skriptu

4.5 Vytvořené programy

Jak již bylo v předchozích částech tohoto textu zmíněno výstupem praktické části této diplomové práce jsou dvě aplikace běžící na platformě .NET. Ty budou popsány v této podkapitole.

Server

Tato část implementace zajišťuje veškerou funkcionalitu, která byla zadavatelem požadována. Bohužel, na základě jeho přání, není možné právě tuto část zveřejnit.

Server pravidelně monitoruje zvolené procesy a v případě, že dojde k některé ze stanovených událostí, provede definovanou akci. Logování věškerých parametrů, kromě výpisu zásobníku probíhá do téhož souboru. Struktura tohoto bude popsána v některé z dalších podkapitol.

Pokud by měl být zaznamenán stav zásobníku, je tento, na přání zadavatele, uložen do zvláštního souboru. Ve zvláštním adresáři na serveru se pak nachází skripty. Jedná se o dávkové soubory systému Windows s příponou *.bat*, ve kterých jsou definovány akce, které je třeba v dané situaci provést.

Windows service také při svém spuštění načítá konfigurace monitorování jednotlivých procesů ze souboru a při svém ukončování do něho aktuální konfigurace ukládá.

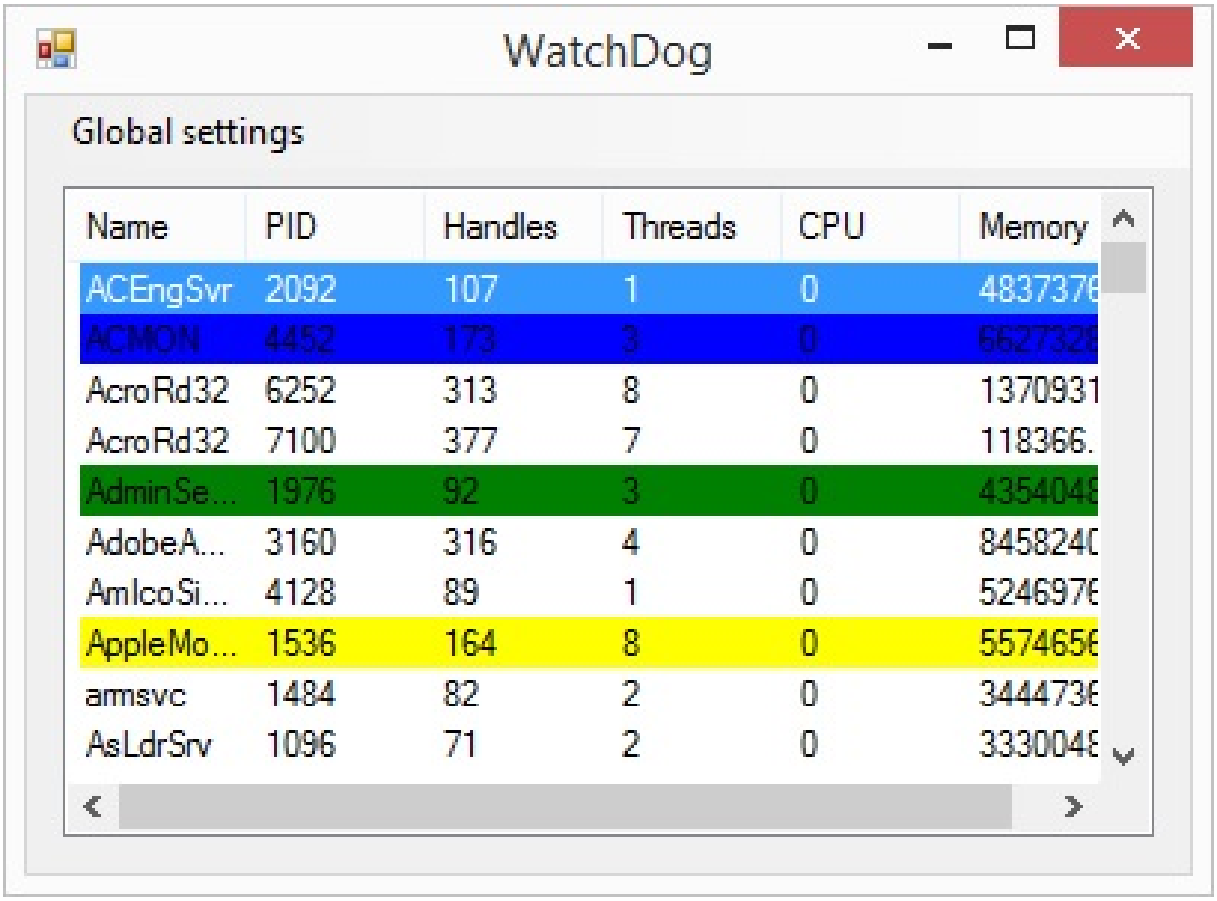
Server pravidelně kontroluje jména a PID všech běžících procesů. V případě, že se objeví nový proces, pro jehož jméno je na serveru uložené nastavení monitorování, začne server automaticky sledovat stanovené hodnoty pro tento proces. Naopak pokud monitorovaný proces skončí, server ho sledovat přestane a to do té doby, než je znovu spuštěn proces se shodným jménem.

Klient

Tato aplikace zprostředkovává uživateli reprezentaci právě běžících procesů a umožňuje mu, aby podle svého přání měnil nastavení monitorování jednotlivých procesu a aby monitorování spouštěl a zastavoval.

Vzhled jednotlivých oken této aplikace byl dohodnut na samostatné konzultaci se zadavatelem a řídí se tak jeho přáními.

Hlavní okno



Obrázek 4.1: Hlavní okno klientské aplikace.

V zobrazovaném okně jsou vypsány všechny procesy, které běží na počítači, na kterém je spuštěna aplikace serveru. Na obrázku je vidět, že jednotlivé procesy mohou být podbarveny různou barvou.

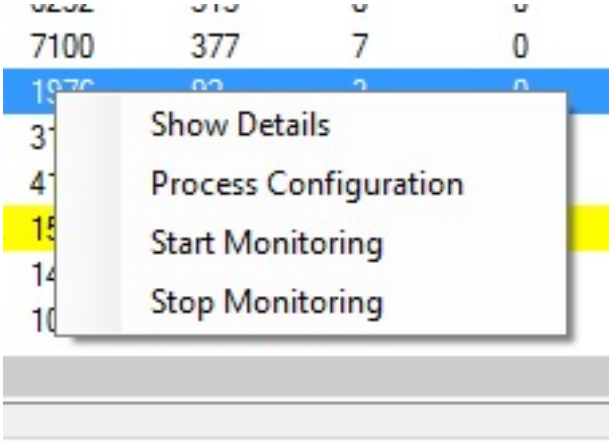
- **Bílá** Proces není monitorován, ani nemá na serveru uložené žádné nastavení, které by spuštění monitorování umožnilo.
- **Modrá** Proces není monitorován, ale na serveru je uložené nastavení, které monitorování umožňuje.
- **Zelená** Proces je monitorován a v poslední době nenastala žádná událost, na kterou mělo být reagováno.
- **Žlutá, Oranžová, Červená** Proces je monitorován a nedávno nastala některá z událostí, na které má býti reagováno. Barva značí závažnost zachycené události.

Toto hlavní okno aplikace je pravidelně aktualizováno v sekundových intervalech, čímž je zajišťována relativní čerstvost zobrazovaných informací a vzhledem k tomu, že interval není příliš krátký, nedochází zasíláním neustálých požadavku k výraznému ovlivnění aplikace, která pracuje v roli serveru.

Bylo zmíněno, že v současné verzi programu, je obarvení procesu pouze dočasné. To je způsobeno tím, že aplikace serveru postupně snižuje čítač, který je nastaven při výskytu události. Pokud tento dosáhne nulové hodnoty, je ze serveru informace o tom, že nastala tato událost smazána. (Pokud mělo proběhnout logování, tak je záznam o události samozřejmě stále uložen v logovacím souboru.) Toto chování bylo implementováno podle přání zadavatele.

Ten však na poslední konzultaci, která proběhla před termínem odevzdání této práce svůj postoj změnil a nově si přeje, aby záznam o proběhlé události z paměti serveru mazán nebyl. To by mělo umožnit vizuální kontrolu toho, zda určité události nastaly či nikoli i po uplynutí delšího časového intervalu. Také má být do hlavního okna aplikace klienta přidáno tlačítko, které zprostředkuje smazání uložených událostí ze serveru a tím umožní kontrolu nových událostí za určitý časový úsek.

Další funkcí, kterou zadavatel od hlavního okna v budoucnosti požaduje je filtrace zobrazovaných procesů podle závažnosti události, která nastala při jejich běhu.



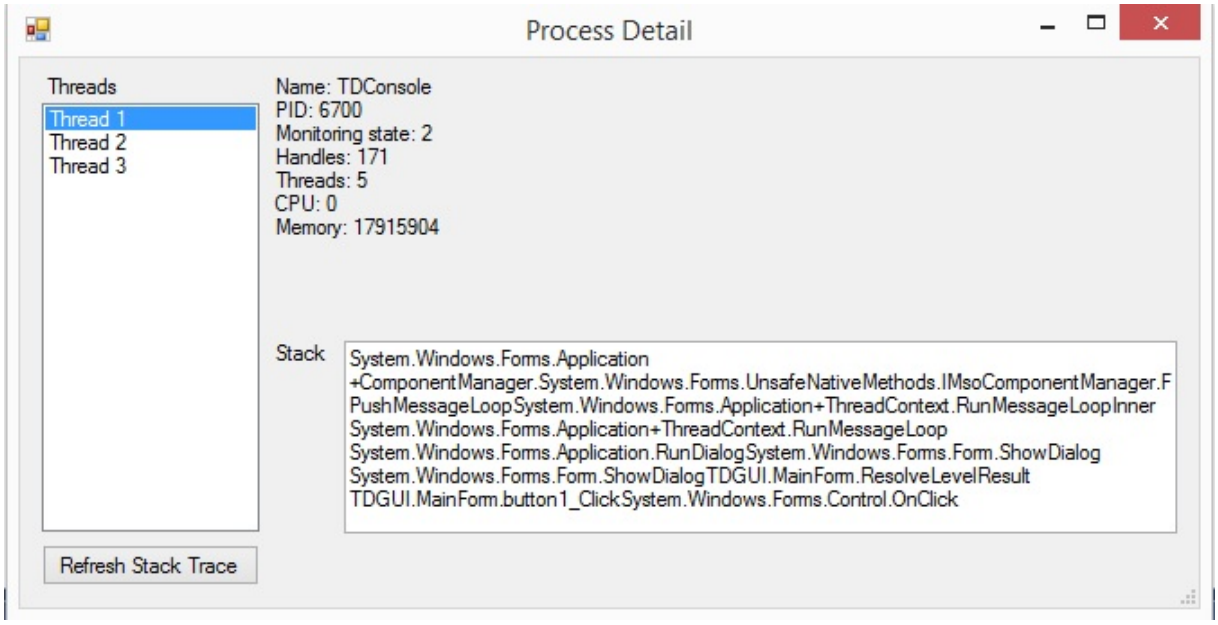
Obrázek 4.2: Kontextové menu.

Uživatel může kliknutím pravým tlačítkem vyvolat kontextové menu, které mu nabízí možnosti akcí, které může se zvoleným procesem provést.

Mezi tyto akce patří spuštění nebo ukončení monitorování zvoleného procesu, výpis detailních informací o tomto nebo nastavení parametrů monitorování.

Detaily o procesu

Ve výpisu detailních informací o zvoleném procesu jsou zobrazeny všechny parametry, jejichž hodnota může být monitorována. Navíc, pokud tento proces běží na platformě .NET, je možno přistoupit k obsahu zásobníku jednotlivých vláken tohoto procesu.



Obrázek 4.3: Detailní informace o zvoleném procesu.

Nastavení monitorování

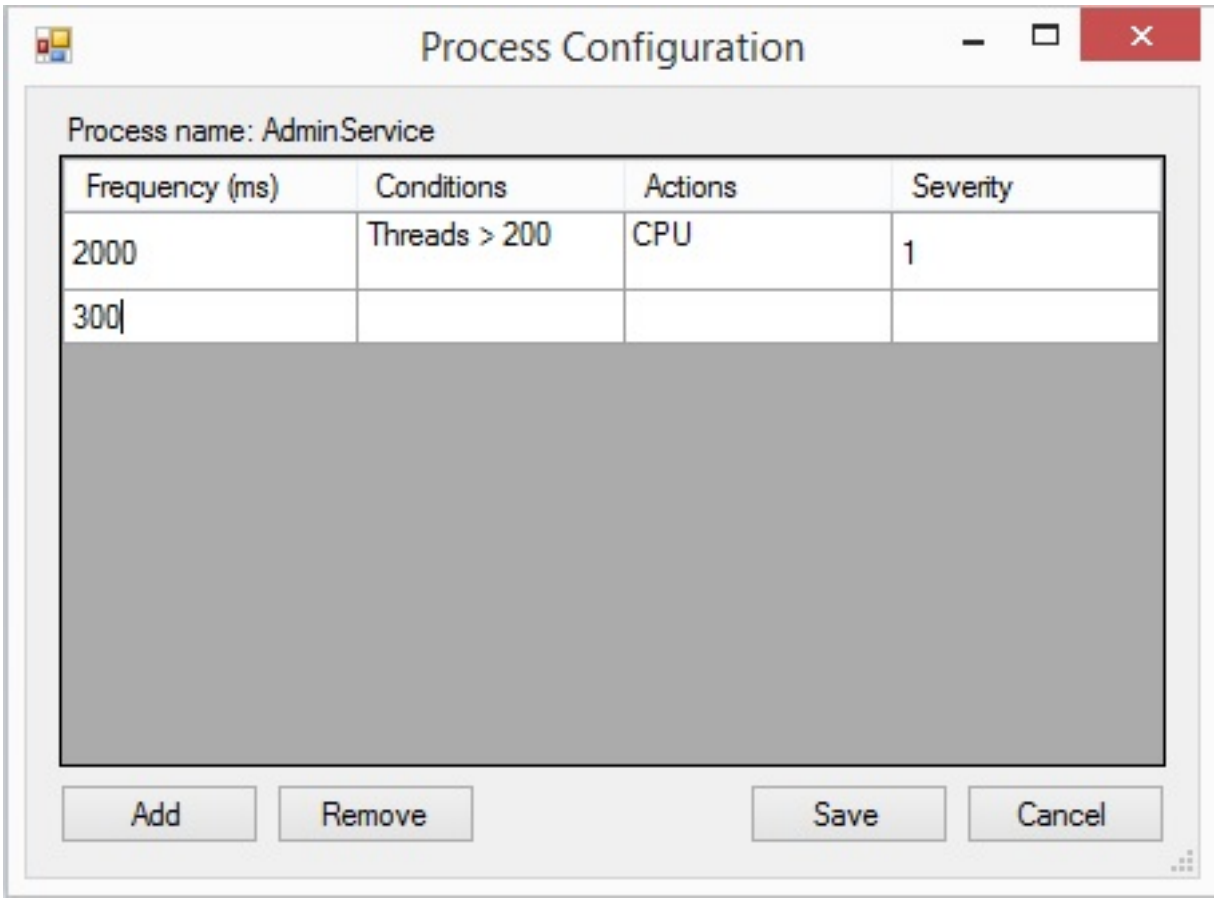
Pokud je v kontextovém menu kliknuto na položku *Process Configuration*, zobrazí se okno pomocí kterého je možno nastavit parametry monitorování procesu s daným jménem.

Tlačítka *Add* a *Remove* umožňují přidávat, respektive odebírat, jednotlivé události, na které má být reagováno. U každé události pak je možno nastavit interval, který určuje kolik milisekund uplyne mezi dvěma kontrolami toho, zda tato událost nastala nebo nikoli.

Je zde také možné nastavit závažnost jednotlivých událostí a to na škále od nuly do dvou. Také zde lze definovat jaké parametry jsou kontrolovány a jaké akce mají být provedeny v případě, že by tyto parametry dosáhly stanovených hodnot. Po skončení úprav je možné provedené změny stiskem tlačítka *Save* uložit. Toto uložení proběhne tak, že na server bude zaslána zpráva (zavolána metoda), která obsahuje nově nastavené parametry monitorování.

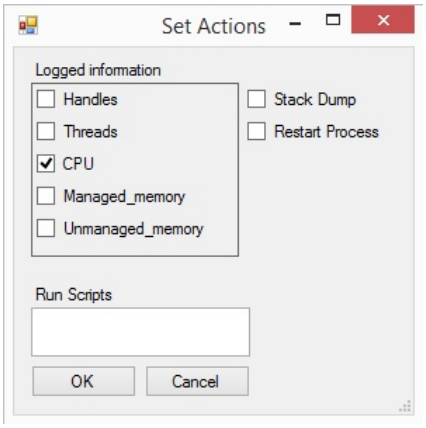
Samotné nastavování podmínek opět probíhá ve zvláštním okně. Parametr, který se má monitorovat a logický operátor podmínky jsou nastavované za pomoci *Combo Box*ů, což zabraňuje zadání neplatných hodnot do těchto polí. Je také možné zvolit konstantu se kterou se bude hodnota zvoleného parametru srovnávat.

Stejně tak má vlastní okno i nastavení akcí, které mají být provedeny, pokud parametry procesu dosáhnou nastavených hodnot. Je možné si vybrat, jaké informace mají být ukládány a zda mají být prováděny nějaké další akce.



Obrázek 4.4: Konfigurační okno.

(a) Nastavení podmínek.



(b) Nastavení akcí.

Obrázek 4.5: Podrobnější nastavení události.

4.6 Související soubory

Serverová aplikaci vytváří při své činnosti soubory již z podstaty její činnosti, kterou je logovaní vybraných parametrů zvolených procesů. Kromě tohoto pracuje při své činnosti i s některými dalšími soubory.

Konfigurační soubor

Jedná se o soubor ve formátu XML, ve kterém jsou uloženy parametry monitorování pro jednotlivé procesy. Při startu Windows služby, jsou z tohoto tyto konfigurace automaticky načteny. Naopak při ukončení služby jsou do souboru ukládány. Příklad, kdy jsou uložena nastavení pro jeden proces je uveden níže.

```
<?xml version="1.0" encoding="UTF-8"?>
<processes>
  <process name="conhost" state="1">
    <event frequency="1000" severity="1">
      <condition text="Threads &gt; 3" />
      <action text="Handles" />
      <action text="Threads" />
    </event>
    <event frequency="2000" severity="0">
      <condition text="Handles &gt; 50" />
      <action text="CPU" />
    </event>
  </process>
</processes>
```

Ze zmíněné ukázky je vidět, že je ukládáno jméno procesu, který má být monitorován, a stav monitorování tohoto procesu. Díky této informaci by aplikace serveru měla být schopná okamžitě po svém startu začít monitorovat procesy, které mají být monitorovány a to bez zásahu uživatele.

Pro každý proces je také uložena skupina událostí, které jsou pro tento monitorovány. Každá událost má stanovenu frekvenci, ve které se kontroluje, zda došlo ke splnění podmínek. Následuje seznam podmínek, jejichž splnění je sledováno a za těmito jsou vypsány akce, které mají být provedeny v případě, že jsou všechny tyto podmínky splněny.

K vytvoření a načtení tohoto souboru je používána třída programovacího jazyka C# *XmlDocument*. Tato zjednodušuje práci se soubory ve formátu XML. Lze si také povšimnout toho, že v dokumentu jsou nahrazeny řídící znaky jazyka XML (konkrétně < a >) sekvencí jiných symbolů.

Logovací soubor

Do tohoto souboru se zapisují parametry, které mají být zalogovány v případě, že by monitorovaný proces dosáhl stanovených parametrů. Příklad zápisu je vidět níže.

```
<?xml version="1.0" encoding="UTF-8"?>
<log>
  <event process_name = "TDConsole" pid = "1440" severity = "1"
    time = "2015-12-06 13:05:16.878" Handles = "173"
    Threads = "6" />
  <event process_name = "Ath_CoexAgent" pid = "3008" severity = "2"
    time = "2015-12-06 14:13:20.142" Threads = "5" />
  <event process_name = "AppleMobileDeviceService" pid = "1536"
    severity = "0" time = "2015-12-07 20:10:40.526" Threads = "8" />
</log>
```

Jednotlivé události tvoří samostatné elementy v tomto dokumentu. Lze do něho zaznamenávat všechny monitorované parametry s výjímkou výpisu stavu zásobníku monitorovaného procesu.

V současné době je serverovou aplikací vytvářen pouze jeden logovací soubor, na jehož konec jsou v případě jejich výskytu přidávány nové události. Tento formát byl zvolen z důvodu požadavku zadavatele. Ten však změnil své preference a v přístí verzi programu má být logování upravno tak, že při startu monitorování bude vytvořen zvláštní soubor, do kterého se budou zaznamenávat pouze události související s procesy s určitým jménem. Tento soubor bude ukončen, pokud bude zastaveno monitorování tohoto procesu.

Tento přístup byl nově zvolen z důvodu, že neustálé logovaní do stejného souboru by vedlo k jeho rychlému růstu a tím náročnější prohledatelnosti. Bylo tak zvažováno řešení, že bude vytvářen nový logovací soubor pro každý den, kdy nějaké logování probíhá. Tento postup byl odmítnut z toho důvodu, že pokud by probíhalo monitorování určitého procesu po delší dobu, bylo by nutné průběh výskytu jednotlivých událostí sestavovat z obsahu více různých souborů. Nově zvolený přístup by měl oba tyto problémy eliminovat.

Výpis zásobníku

Dalším typem souborů, které přímo souvisí s činností serveru jsou soubory, do kterých se ukládá výpis obsahu zásobníku monitorovaných procesů. Již bylo zmíněno, že současný stav aplikace umožňuje tato data získat pouze pro procesy, které běží na platformě .NET, vzhledem k tomu, že o monitorování přesně tohoto typu procesů má zadavatel diplomové práce zájem, nebylo toto shledáno jako problém.

Podle přání zadavatele je obsah zásobníku ukládán zvlášť a nezapisuje se se do obecného logovacího souboru. Pro každý výpis je vytvořen nový textový soubor, který je umístěn ve složce, jejíž označení je shodné se jménem procesu, jehož výpisy jsou do

ní ukládány. Soubor pak svůj název dostává podle data a času, ve kterém byl výpis zásobníku pořízen. Obsah vzorového souboru je možné vidět níže.

```
THREAD 1
System.Windows.Forms.Application+ComponentManager.System.Windows.
Forms.UnsafeNativeMethods.IMsoComponentManager.FPushMessageLoop
System.Windows.Forms.Application+ThreadContext.RunMessageLoopInner
System.Windows.Forms.Application+ThreadContext.RunMessageLoop
System.Windows.Forms.Application.RunDialog
System.Windows.Forms.Form.ShowDialog
System.Windows.Forms.Form.ShowDialog
TDGUI.MainForm.ResolveLevelResult
TDGUI.MainForm.button1_Click
System.Windows.Forms.Control.OnClick
System.Windows.Forms.Button.OnClick
System.Windows.Forms.Button.OnMouseUp
System.Windows.Forms.Control.WmMouseUp
System.Windows.Forms.Control.WndProc
System.Windows.Forms.ButtonBase.WndProc
System.Windows.Forms.Button.WndProc
System.Windows.Forms.Control+ControlNativeWindow.OnMessage
System.Windows.Forms.Control+ControlNativeWindow.WndProc
System.Windows.Forms.NativeWindow.Callback
System.Windows.Forms.Application+ComponentManager.System.Windows.
Forms.UnsafeNativeMethods.IMsoComponentManager.FPushMessageLoop
System.Windows.Forms.Application+ThreadContext.RunMessageLoopInner
System.Windows.Forms.Application+ThreadContext.RunMessageLoop
System.Windows.Forms.Application.Run
TowerDefence.Program.Main

THREAD 2

THREAD 3
Microsoft.Win32.SystemEvents.WindowThreadProc
System.Threading.ThreadHelper.ThreadStart_Context
System.Threading.ExecutionContext.RunInternal
System.Threading.ExecutionContext.Run
System.Threading.ExecutionContext.Run
System.Threading.ThreadHelper.ThreadStart
```

Program, jehož zásobník byl zkoumán byl vytvořen v programovacím jazyce C# za po použití Windows Forms. Z metod, které jsou vypsány je vidět, že obsah zásobníku skutečně odpovídá takovémuto typu programu.

Takovýto formát výstupu je ve shodě s potřebami zadavatele. Použitá knihovna

MdbgEngine umožnuje zásobník zkoumat do větších podrobností, a tak je v případě požadavku zadavatele možné vypsat dodatečné informace o jeho obsahu.

Skripty

V tomto případě nebude popsán žádný ukázkový soubor. Neboť je možné, aby na tomto místě vystupoval jakýkoli dávkový soubor.

Tyto soubory jsou uloženy ve zvláštní složce na počítači, na kterém je spuštěna aplikace serveru. Uživatel může prostřednictvím grafického uživatelského rozhraní, při volbě akcí, které mají být provedeny, pokud monitorovaný proces dosáhne zvolených parametrů, nastavit jméno souboru se skriptem, který má být v takové situaci spuštěn.

Kapitola 5

Závěr

Výstupem této diplomové práce je nejen tento text, ale také dva programy vytvořené za pomoci jazyka C#. Tyto jsou schopny při vzájemné spolupráci poskytnou funkcionalitu, která byla požadována zadavatelem práce a která je popsaná v předchozích kapitolách této práce.

Výsledný produkt není zatím ve společnosti SEACOMP s.r.o. používán, nicméně byl z jejich strany vyjádřen zájem o pokračování jeho vývoje.

Jako kandidát na další rozšíření funkcí programu je navrženo, že by tento měl být v budoucnosti schopný předat uložená naměřená data ze serveru klientovi, který by tato dokázal zobrazit. Nejlépe v grafické podobě.

Literatura

[1] Joseph Albahari, Ben Albahari. *C# 4.0 in a Nutshell, Fourth Edition* . O'Reilly Media, 2010.

[2] *Introduction to the C# Language and the .NET Framework.* Microsoft Corporation. [online], 2013 [cit. 6. 12. 2014]. Dostupné na: http://msdn.microsoft.com/en-au/library/z1zx9t92.aspx.

[3] Pablo Cibraro, Kurt Claeys, Fabio Cozzolina, Johann Grabner. *Professional WCF 4: Windows Communication Foundation with .NET 4.* Wiley Publishing, 2010.

[4] Nishith Pathak. *Pro WCF 4: Practical Microsoft SOA Implementation, Second Edition.* Apress, 2011.

[5] Abhishek Sur. *Visual Studio 2013 and .NET 4.5 Expert Cookbook.* Packt Publishing, 2014.

[6] *Endpoints: Addresses, Bindings, and Contracts.* Microsoft Corporation. [online], [cit. 8. 12. 2014]. Dostupné na: http://msdn.microsoft.com/en-us/library/ms733107

[7] *Fault Contract.* Microsoft Corporation. [online], [cit. 8. 12. 2014]. Dostupné na: http://msdn.microsoft.com/cs-cz/library/ms752208

[8] Joe Fawcett, Liam R.E. Quin, Danny Ayers. *Beginning XML.* Wiley Sons, 2012.

[9] International Organization for Standardization. *ISO 8879:1986.* [online], 2008 [cit. 13. 12. 2014]. Dostupné na: http://www.iso.org/iso/catalogue$_detail.htm?csnumber = 16387.$

[10] W3 Consorcium. *Extensible Markup Language (XML).* [online], [cit. 13. 12. 2014]. Dostupné na: http://www.w3.org/XML/.

[11] W3 Consorcium. *Extensible Markup Language (XML) 1.0 (Fifth Edition).* [online], 2008 [cit. 13. 12. 2014]. Dostupné na: http://www.w3.org/TR/REC-xml/.

[12] *What is Task Manager?* Microsoft Corporation. [online], [cit. 14. 12. 2014]. Dostupné na: http://windows.microsoft.com/en-us/windows-vista/what-is-task-manager.

[13] Sasha Goldshtein. *CLR Stack Explorer – Preview*. [online], 2011 [cit. 21. 12. 2014]. Dostupné na: http://blogs.microsoft.co.il/sasha/2011/07/19/clr-stack-explorer-preview/.

Přílohy

Přílohou této práce je prohlášení zaměstnace zadavatele pověřeného průběžnými kontrolami postupu vypracování praktické části této práce. Toto je dostupné v archivu této diplomové práce v Informačním systému Masarykovi univerzity. Zdrojové kódy samotné naopak součástí přílohy nejsou, neboť zadavatel si nepřál, aby byly tyto zveřejněny.