Benchmarking, sběr a vizualizace dat v systému AVG Technologies

Benchmarking, gathering and data visualisation system for AVG Technologies

Bc. David Rohlík

Diplomová práce
2010

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2009/2010

ZADÁNÍ DIPLOMOVÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. David ROHLÍK**
Osobní číslo:	**A07653**
Studijní program:	**N 3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Téma práce:	**Benchmarking, sběr a vizualizace dat v systému AVG Technologies**

Zásady pro vypracování:

1. **Vytvořte literární rešerši na téma benchmarking softwarových aplikací a vizualizace dat a zhodnoťte současný stav elektronických systému určených k řešení této problematiky.**
2. **Navrhněte a vytvořte softwarový systém pro sběr, benchmarking a vizualizaci dat pořízených při zátěžových a výkonnostních testech SW komponent společnosti AVG Technologies, jakými jsou např. Rezidentní štít, Webový štít a Firewall.**
3. **Součástí práce bude také softwarová aplikace, která určí dle jakých kritérií se budou zpracovávat datové logovací soubory vytvářené při výše zmíněných testech a ty poté požadovaným způsobem vizualizovat, zapisovat do tabulek, případně do HTML výstupu.**
4. **Vytvořte programovou dokumentaci Vašeho systému.**

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **Benchmark_(computing): Wikipedie otevřená encyklopedie [online]. 2009. Dostupný z WWW: http://en.wikipedia.org**
2. **Brown, A. B. A Decompositional Approach to Computer System Performance Evaluation. Center for Research in Computing Technology, Harvard University, 1997**
3. **Dongarra, Jack J. Computer Benchmarks (Advances in Parallel Computing). North-Holland, 1993. 364 s. ISBN 978-0444815187**
4. **Grace, R. The Benchmark Book. Prentice Hall, 1996. 313 s. ISBN 978-0133418019**
5. **Dujmovic, J. J., Howard, L. A Method for Generating Benchmark Programs, Department of Computer Science, San Francisco State University, 2000.**
6. **Hoffman, E. Patrick. Table vizualizations: A Formal Model and Its Application. University of Massachusetts, 1999. 239 s. Dostupné z WWW: http://www.cs.uml.edu/ phoffman**
7. **Electronic Statistics Textbook [online]. 2008. Dostupný z WWW: http://www.statsoft.com/textbook**
8. **Fry, Ben. Visualizing Data. OReilly Media, Inc. 2008. ISBN 978-0-596-51455-6**

Vedoucí diplomové práce: **Ing. Michal Bližňák, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **19. února 2010**

Termín odevzdání diplomové práce: **8. června 2010**

Ve Zlíně dne 19. února 2010

prof. Ing. Vladimír Vašek, CSc.
děkan

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

prof. Ing. Vladimír Vašek, CSc.
ředitel ústavu

ABSTRAKT

Cílem práce je vytvoření vhodného systému pro spuštění testů, které budou schopny naměřit data, jež budou odeslány do centrálního úložiště, kde budou dále zpracovány a provedena jejich následná vizualizace. V průběhu textu jsou představeny technologie, které byly pro tento systém využity. V textu je též představena problematika benchmarkingu a je uvedeno několik současných systému, které jsou pro podobné účely využívány.

Systém je navržen tak, aby nebyl využitelný pouze pro jeden konkrétní účel jakým je benchmarking. Systém umožňuje spuštění automatických skriptů pro automatické testování, jehož výsledky je možné využít pro QA oddělení softwarových společností.

Klíčová slova:

benchmarking, automatizace, ASP.NET, C#, Powershell, klient server aplikace

ABSTRACT

The goal of this work is to create an appropriate system to start tests which will be able to measure performance and testing data. This data will be sent to the central repository, where it will be processed and will be used for the data visualization. In the text, there is benchmarking dilemmas introduced together with some current systems intended for similar purposes.

My system is projected to be used for more purposes than benchmarking only. The system is able to start automatic scripts for testing automatization. It’s results can be used in the QA departments of the software vendors.

Keywords:

benchmarking, automatization, ASP.NET, C#, Powershell, client server application

Poděkování

Velmi rád bych poděkoval a vyslovil uznání všem, kteří mi pomáhali při vzniku této práce. Především Ing. Michalu Bližňákovi, Ph.D., vedoucímu mé diplomové práce za trpělivé vedení.

Nakonec bych chtěl poděkovat rodičům za poskytnuté zázemí a své přítelkyni za její trpělivost a lásku.

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

Diplomová práce pojednává o problému vytvoření systému pro management automatického testování, sběru dat z testovacích stanic a vizualizaci dat obsahujících i grafické zobrazení dat používané pro benchmarking.

Na úvod práce se zabývám tématikou benchmarkingu a současných řešení pro měření výkonu. První kapitoly práce obsahují i představení použitých technologií ASP.NET, C# a Powershell.

V dalších kapitolách je rozebrán návrh systému a řešení aplikace. Snažil jsem se popsat jak způsob, kterým jsem aplikaci vytvořil, tak pohled z uživatelského hlediska a využití systému pro běžné uživatele.

Na úplný závěr jsou shrnuty možnosti aplikace, její nedostatky a možné rozšíření do budoucna.

I. TEORETICKÁ ČÁST

1 NASTÍNĚNÍ PROBLEMATIKY A POUŽITÉ TECHNOLOGIE

1.1 Benchmark

1.1.1 Benchmark obecně

Benchmarking je technika pro experimentální měření výkonu, která se pokouší změřit výkon reálně vytíženého systému. Výkon systému je měřen specializovanou aplikací, nazývanou benchmark, která je vytvořená simulovat reálné vytížení systému. Benchmarking pomáhá zodpovědět základní otázku, zda systém splňuje či nesplňuje požadavky na výkon. Pokud taková aplikace neodpovídá požadavkům, je třeba odladit její parametry nebo formu implementace.

Benchmark je definován jako sada programů (nebo mikro programů), jež jsou spouštěny na různých systémech s cílem změřit jejich výkon. Výsledky standardních benchmarků reflektují pouze výkon množiny programů na měřeném systému. V reálu je ale velký rozdíl mezi programy zahrnutými ve standardních benchmarcích a reálnými aplikacemi. Následkem pak mohou být desinformační a matoucí výsledky měření.

Velice často je termín benchmarking spojován s měřením výkonu počítačového hardware. Například měření výkonu CPU, grafických čipů nebo rychlostí zápisu či čtení pevných disků. Používán je však také ve spojení se software, kde se dají využít i techniky pro měření výkonu hardware. Mezi softwarové benchmarky se řadí například testy proti kompilerům nebo měření výkonu databázových systémů. Do softwarových benchmarků spadají i testy pro vyhodnocení korektnosti software.

1.1.2 Účel počítačového benchmarkingu

Jak se počítačová architektura vyvíjela, bylo velmi těžké porovnat výkonnost různých počítačových systémů pouze podle jejich specifikace. Proto byly vyvinuty testy, které umožnily porovnání různých architektur. Jako příklad může sloužit měření výkonu procesorů Pentium 4 a Athlon XP, kde procesor Pentium 4 běží na vyšší frekvenci, což ovšem nemusí znamenat větší výkon.

Benchmarky jsou vytvořeny k simulaci zátěže komponenty nebo systému. Syntetické benchmarky simulaci provádí pomocí speciálně vytvořených programů, které simulují

zátěž specifické komponenty. Aplikační benchmarky naopak spouští reálné programy na testovaném systému. Zatímco aplikační benchmarky obvykle vyjadřují mnohem lepší výsledky z reálného běhu na daném systému, syntetické benchmarky jsou dobré pro test konkrétních komponent, jako je pevný disk nebo pro rychlost vyhledávání v databázi.

Benchmarking je důležitý pro vývoj procesorů. Dává architektům možnost měřit a následně vytvářet kompromisy při vytváření mikroarchitektury procesorů. Takto je možné spouštět různé programy na otestování klíčových algoritmů a na základě jejich výsledků je možné se rozhodnout, které řešení je vyhovující. Případně může takový test vést i k nalezení jiné cesty pro zlepšení výkonu.

Nalezení správných parametrů nebo změn v implementaci vedoucích ke zlepšení výkonu ovšem může být velice složité. Optimální hodnoty parametrů mohou být nalezeny spouštěním benchmarkingu pro každou možnou kombinaci parametrů. Pokud je počet kombinací příliš velký a nebo parametry nejsou na sobě závislé, optimální výkon je možné najít i bez nutnosti testovat všechny kombinace. Výběr kombinací parametrů a jejich hodnot pro otestování může být založeno na teorii experimentálního designu [Jain, 1991]

V prostředí komerčních benchmarků jsou ovšem výrobci počítačů a software známi tím, že konfigurují jejich systémy tak, aby dávaly nerealistické výsledky, které nemohou být replikovány v reálném světě. Například během 80.tých let minulého století některé kompilery detekovaly specifickou matematickou operaci, která byla využívána v dobře známém benchmarku s plovoucí čárkou. Po nalezení této operace ji nahradili jinou, rychlejší, metodou, která dávala matematicky ekvivalentní výsledek. Ve většině případů lze konstatovat, že výrobci jak software, tak hardware prezentují pouze takové benchmarky, které ukazují jejich produkty v tom nejlepším světle.

1.1.3 Kvalita současných benchmarkingů

Benchmarking obvykle obsahuje několik iterativních kol měření tak, aby byly výsledky co nejpoužitelnější. Interpretace dat naměřených benchmarkingem je taktéž obtížná disciplína.

V roce 1988 vznikla nezisková organizace Transaction Processing Performance Council (TPC). Tato organizace definovala benchmarky pro transakční zpracování, databáze tak, aby data naměřená definovanými benchmarky byla důvěryhodná. TPC benchmarky jsou

dnes rozšířené pro ohodnocení výkonu počítačových systému a jejich výsledky jsou zveřejněny na webu společnosti TPC.

Bohužel nejvíce dostupné jsou výsledky benchmarků, které jsou naměřeny na reklamu daného produktu nebo amatérské benchmarky sledující pouze jednu vlastnost, případně benchmarky spuštěné nad nesprávnými daty.

Pro příklad uvádím několik běžných problémů, se kterými se lze ve světě počítačového benchmarkingu setkat:

- Vývojáři obvykle vylepšují své produkty tak, aby obstály v tzv. standardizovaných benchmarkigových testech. Pro příklad lze uvést produkt Norton SysInfo, kde je relativně jednoduché vyladit produkty tak, aby bez problémů prošly prováděnými testy
- Výrobci software a samotní vývojáři bývají nařčeni, že výsledky bechmarku, které sami zveřejňují, falšují
- Mnoho benchmarků se plně soustředí pouze na rychlost a úplně zanedbává ostatní důležité vlastnosti jako například:
    - Kvalita služeb – tento termín obsahuje vlastnosti jako bezpečnost, dostupnost, spolehlivost, rozšiřitelnost. Pro kompletní ohodnocení systému, do kterého se počítají i tyto vlastnosti, byla navržena metoda ACID (atomicity, consistency, isolation, durability). Metoda ACID byla popsána [3].
    - Obecně výsledky benchmarků nevyjadřují hodnotu zvanou Total cost of ownership. Ta zastupuje finanční ohodnocení produktu a ukazuje přímou a nepřímou cenu produktu nebo systému. Organizace TCC definovala speciální metriky, které se vyjádřením ekonomických hledisek zabývají. Poprvé byla přímá a nepřímá cena produktu definována společností Gartner Group v roce 1987.
    - Spotřeba elektrické energie. Pokud je zapotřebí větší spotřeby energie, přenosné systémy (jako např. notebooky) budou mít menší výdrž baterie a bude nutné je častěji dobíjet. Problém se netýká pouze hodnocení výkonu hardwarových komponent, ale i software. Pokud bude software permanentně zatěžovat systém, spotřeba elektrické energie bude nesrovnatelně vyšší v porovnání se systémem v klidovém stavu
    - Kvalita kódu a jeho délka. Pokud bychom vzali v potaz systémy pro přenosné počítače nebo mobilní zařízení, velkou roli bude hrát zabraná paměť nebo místo na disku, či jiném úložišti. Větší hustota kódu může tento problém řešit

- Výsledky benchmarků zveřejňovaných výrobci většinou ignorují požadavky pro vývoj, testování a zotavení z pádu. Udržují tak pořizovací cenu produktu na nejnižší možné míře. Kupříkladu k novému databázovému systému je nutné věnovat čas k nasazení, testování a posléze také dokoupit zálohovací zařízení.
- Benchmarky mají problém se přizpůsobit distribuovaným serverům. Speciálně těm s vysokou citlivostí na síťovou topologii. Do takových sítí spadají například sítě typu Grid computing, kde na řešení jednoho problému spolupracuje více serverů či pracovních stanic. V těchto případech je velice obtížné vyhodnocovat výsledky naměřených dat.
- Uživatelé mohou mít úplně jiné vnímání výkonu, než jak jej zobrazují benchmarky. Obecně uživatelé oceňují předvídatelnost, kdy kupříkladu servery vždy splní dané požadavky včas nebo je splní dříve, nežli je očekáváno.
- Výkon mnoha serverových architektur dramaticky poklesne, pokud dojde k vytížení blížícímu se 100%. Benchmarky ale nemusí s touto situací počítat. Výrobci někdy zveřejňují serverové benchmarky při vytížení kolem 80%, což dává zkreslené výsledky
- Instituce zabývající se benchmarkingem často neberou na vědomí základní vědecké postupy. To obnáší například malou sbírku vzorků pro sledovanou činnost nebo omezenou možnost opakování naměřených hodnot

1.1.4 Typy benchmarků

1. Benchmarking reálných programů.
   - Do této kategorie spadá například komplexní benchmarking textových procesorů, antivirového software apod.
2. Kernel benchmarking
   - Sem patří například LINPACK Benchmarks. Jedná se o benchmark zaměřený na výkon systému při počtech s plovoucí čárkou. Výsledky jsou prezentovány pomocí jednotek MFLOPS (počet miliónů operací v plovoucí řádové čárce za sekundu). LINPACK benchmarky jsou blíže popsány [4].
3. Component benchmark
   - Zahrnuje programy určené k měření výkonu základních počítačových komponent
4. Syntetický benchmark. Ten se dále dělí na:
   a. Whetstone benchmark [12]
   b. Dhrystone benchmark [12]

- Rozdíly mezi nimi jsou především ve faktu, že Dhrystone benchmark nepoužívá žádné operace s plovoucí čárkou, kdežto Whetstone je obsahuje. Výsledkem benchmarků jsou data udávaná v počtu Dhrystones nebo Whetstones

5. I/O benchmarks
   - Měření se týká počtu vstupních případně výstupních operací, které je schopen hardware nebo software zpracovávat
6. Paralelní benchmark
   - Jsou využívány na počítačích s více procesory nebo na systémech, které obsahují několik počítačů

1.1.5 Jak mohou být výsledky benchmarku zavádějící

Jak již bylo zmíněno na začátku, existují velké rozdíly mezi výsledky standardních benchmarků a mezi výkonem reálných aplikací. Může to být demonstrováno na jednoduchém příkladu. Představme si hypotetický scénář, kde systémový administrátor je postaven před úkol koupit nový poštovní server pro svoje oddělení. Předpokládejme, že ze všech existujících řešení jsou vybrány dvě (systém A a systém B), které se pohybují ve stejné cenové hladině. Který z nich bychom si tedy vybrali jako náš budoucí emailový server?

Běžný postup pro takové případy by bylo spuštění standardního obecného benchmarku na zmíněných dvou systémech a poté vybrat ten, který bude mít lepší hodnocení. Alternativní přístup k řešení problému by zahrnoval využití specifičtějších benchmarků, které otestují konkrétní aspekty systému, jež jsou důležité pro výkon vybrané aplikace (emailového serveru). K měření tedy využijeme benchmark sloužící k otestování výkonu CPU a následně benchmark, který je určen k otestování výkonu emailových serverů. Výsledky benchmarku CPU nám ukáží, že oba systémy jsou srovnatelné. Z výsledku benchmarku emailového serveru vidíme, že systém A má lepší výkon. Z testů, které jsme ovšem provedli, zjistíme, že systém B dokáže zvládnout mnohem více emailové komunikace než systém A.

Z uvedeného příkladu nám vyplývá, že standardní benchmarky ne vždy reprezentují chování reálných aplikací. Ačkoli měření výkonu emailových serverů by znamenaly jasnou

volbu pro systém A, systém B dokáže zpracovat mnohem více emailové komunikace. Z toho ovšem také plyne, že ačkoli benchmarky nutně nereprezentují chování aplikací, tak mohou naznačit reálný výkon aplikace [7].

1.1.6 Současná benchmarkingová řešení

SPEC CPU2006

Benchmark vyvinutý společností Spec (Standard Performance Evaluation Corporation). Jedná se o standardizovaný benchmark zaměřený na měření výkonu procesoru, paměti a kompilátoru testovaného systému spouštěním řady úloh a zaznamenáváním času potřebného k jejich dokončení. Obsahuje 2 soubory testů:

- CINT2006 pro testování výpočtů celočíselných operací
- CFP2006 pro testování výpočtů s plovoucí čárkou

CINT2006 obsahuje 12 benchmarků založených na reálných aplikacích napsaných v C a C++. CFP2006 zahrnuje 17 benchmarků napsaných v C, C++, různých verzích Fortranu a C/Fortran.

SPEC CPU2006 vyhodnocuje benchmarky ve dvou výkonnostních hodnoceních nazývaných SPECint2006 a SPECfb2006. Obě tyto vyhodnocení poskytují dvě důležité metriky výkonu systému. První vyhodnocuje, jak rychle dokáže systém vyřešit jednu úlohu. Metrika je nazvána speed (rychlost). Druhé vyhodnocení ukáže kolik je systém schopen splnit úloh za časovou periodu a tato metrika je nazývána výkon (throughput).

Výsledky jsou formátovány pomocí utilit do HTML, CSV, textu, PDF nebo PS skriptu.

WorldBench

Vyvíjený společností PC World Labs od roku 2000. Jedná se o benchmark pro operační systémy Windows, jehož výsledky firma zveřejňuje každý měsíc ve vyhodnocení různých počítačových komponent.

WorldBench využívá pro měření výkonu populární desktopové aplikace, aby dosáhl co největšího přiblížení k reálnému využití. Benchmark nainstaluje speciálně upravené verze testovaných aplikací, které jsou přizpůsobeny, aby jejich úkony šlo lépe automatizovat skripty.

Při spuštění testu jsou spuštěny testovací skripty. Ty obvykle obsahují běžnou práci s aplikacemi jako výběry z menu, používání zkratek a klikání myší v testovaných aplikacích. Testované aplikace jsou přímo dané výrobcem a obsahují např. Adobe Photoshop CS2, Firefox, Microsoft Office, Autodesk 3ds a další.

Výsledky aplikací jsou uvedeny přímo v benchmarkové aplikaci v přehledné tabulce, kde je pro každou testovanou aplikaci zobrazeno dosažené skóre. Tyto výsledky lze poté exportovat do různých formátů.

SYSmark 2007

SYSmark 2007 byl vyvinut společností BABCo v roce 2007 pro operační systémy Windows. SYSmark se stejně jako WorldBench soustředí na výkon předdefinovaných reálných aplikací v oblastech tvorby videa, 3D modelování nebo produktivity v textových nebo tabulkových procesorech. SYSmark je využíván vcelku širokou škálou výrobců hardware a vývojářů.

Samotné testování opět probíhá provedením automatizovaných kroků v reálných aplikacích. Benchmark obsahuje například aplikace Adobe Photoshop, Autodesk 3ds, Macromedia Flash, Microsoft Office nebo WinZip.

Výsledky jsou zobrazeny v přehledové tabulce po doběhnutí kompletního benchmarku. Pro každé odvětví (E-learning, VideoCreation, Productivity a 3D) je zobrazen výsledný počet bodů, jenž lze porovnávat s výsledky na jiných systémech. Lze také vygenerovat grafy výkonu v jednotlivých testovaných aplikacích. Samozřejmostí zůstává možnost exportu do různých formátů.

1.2 ASP.NET

1.2.1 Obecně o ASP.NET a historie

ASP.NET je webový aplikační framework vyvinut společností Microsoft. ASP. NET umožňuje vývojářům vytvořit dynamické webové stránky, webové aplikace nebo webové služby. První verze byla uvolněna roku 2002 společně s verzí 1.0 .NET Frameworku. ASP.NET je postaven na komponentě Common Language Routine (CLR), která dovoluje programátorům napsat jakýkoli kód ASP.NET s využitím jakéhokoli podporovaného jazyku .NET [11].

1.2.2 Použité prvky a jejich vlastnosti

Stránka

Stránka v .NET jsou oficiálně známy jako webové formuláře a jsou základním stavebním kamenem pro vývoj aplikací. Webové formuláře jsou obsaženy v souborech s příponou „.aspx“. Stránky typicky obsahují (X)HTML značky, stejně tak ovládací prvky (Web Controls a User Controls), kde vývojáři umisťují veškerý dynamický a statický obsah webových stránek.

Samotnými tvůrci ASP.NET je doporučeno používat takzvaný „Code-behind“ model tvorby webových stránek. Dynamický kód je poté umístěn do separátního souboru nebo do speciálních skriptovacích značek. ASP.NET je založen na CLR (Common Language Runtime), který je sdílen všemi aplikacemi postavenými na .NET Frameworku. Programátoři tak mohou realizovat své projekty v jakémkoliv jazyce podporujícím CLR, např. Visual Basic.NET, JScript.NET, C#, Managed C++, ale i mutace Perlu, Pythonu a další.. Kód psaný v „Code behind“ modelu reaguje na různé události jako načtení stránky nebo stisk tlačítka, což představuje velký rozdíl oproti procházení kódu od shora dolů [11].

Management stavů

ASP.NET aplikace jsou hostovány pomocí web serveru a přístup k nim je realizován pomocí bezstavového HTTP protokolu. ASP. NET využívá několik funkcí pro management stavů. Mezi ně patří stav aplikace, stav session, View state, cachování na úrovni serveru, cookies a využití parametrů stránky.

Využité technologie v mé diplomové práci jsou View state a parametry stránky, proto se jim budu věnovat detailněji.

View state

Funkce zpracování stránky ve webových aplikacích je založena na přijetí klientova požadavku na zaslání stránky, její vygenerování serverem a odeslání klientovi. Proces generování stránky může mít různé formy - od pouhého přečtení souboru s uloženou statickou HTML stránkou až po vykonání serverových skriptů, které sestaví požadovanou stránku. Ve webové aplikaci je často potřebné udržovat stavové informace mezi jednotlivými požadavky na generování stránek s různým rozsahem platnosti - některé informace jsou platné pro všechny požadavky na aplikaci, některé pouze pro konkrétního uživatele pracujícího s aplikací a podobně.

ViewState je nástroj umožňující uchování stavu stránky a serverových objektů na ní umístěných mezi jejím opakovaným zpracováním. Na rozdíl od jiných metod uchování stavu v ASP.NET aplikacích, které je možno použít v rozsahu platnosti uživatele aplikace (Session) nebo celé aplikace (Application, Cache, ...) je tedy jeho platnost omezena pouze po dobu tohoto opakovaného zpracování jedné stránky.

ViewState je využívána zejména pro uchování hodnot vlastností ovládacích prvků umístěných na stránce, ale lze ji samozřejmě také použít v kódu stránky.

Druhým, již méně často zmiňovaným, ale z hlediska provozovatele aplikace podstatně nepříjemnějším problémem, je zabezpečení takto uložených hodnot. Ve výchozím nastavení stránek je totiž serializovaný řetězec pouze zakódován pomocí Base64 kódování. Formát Base64 je primárně používán ve zprávách elektronické pošty, principielně se jedná o zakódování binárních dat do tisknutelných znaků. Data uložená v tomto formátu jsou však po provedení zpětného dekódování čitelná komukoliv, kdo zakódovaný řetězec získá. Pokud je tedy takto zakódovaný ViewState odeslán na klienta, je možné jej dekódovat nejen v klientově prohlížeči, ale pokud není pro přenos použit šifrovaný HTTPS protokol, také na kterémkoli síťovém prvku, přes který stránka putuje od serveru ke klientovi. Z tohoto důvodu je ukládání jakýchkoli citlivých dat do ViewState značným bezpečnostním rizikem [11].

Query state

Jedná se o starou metodu, jež je součástí URL (Uniform Resource Locator) a která obsahuje data, která jsou předána webovým aplikacím ke zpracování.

Typická URL, která obsahuje zmíněný řetězec je například:

http://server/path/program?query_string

Jakmile server dostane požadavek na stránku, spustí program, kterému předá „query_string“. Otazník je v tomto případě pouze oddělovač od samotné stránky a není součástí parametrů.

1.3 Programovací jazyk C#

Pomocí jazyka C# jsou psány v mé diplomové práci tzv. „Code-behind“ sekce ve webové aplikaci pro spouštění a management testů. Jazyk C# je také použit pro aplikaci, jenž funguje v pozadí webové aplikace a to vícevláknový server-klient, jenž jednotlivé testy spouští a sbírá z nich data [10].

1.3.1 Obecně o C# a jeho historie

C# je vysokoúrovňový objektově orientovaný programovací jazyk vyvinutý firmou Microsoft zároveň s platformou .NET Framework, později schválený standardizačními komisemi ECMA (ECMA-334) a ISO (ISO/IEC 23270). Microsoft založil C# na jazycích C++ a Java (a je tedy nepřímým potomkem jazyka C, ze kterého čerpá syntaxi) [10].

C# lze využít k tvorbě databázových programů, webových aplikací a stránek, webových služeb, formulářových aplikací ve Windows, softwaru pro mobilní zařízení (PDA a mobilní telefony).

Standard ECMA definuje současný design C# takto:

- C# je jednoduchý, moderní, mnohoúčelový a objektově orientovaný programovací jazyk.
- Jazyk a jeho implementace poskytuje podporu pro principy softwarového inženýrství, jako jsou: hlídání hranic polí, detekce použití neinicializovaných proměnných a automatický garbage collector. Důležité jsou také jeho vlastnosti jako: robustnost, trvanlivost a programátorská produktivita.
- Jazyk je vhodný pro vývoj softwarových komponent distribuovaných v různých prostředích.
- Přenositelnost zdrojového kódu je velmi důležitá, obzvláště pro ty programátory, kteří jsou obeznámeni s C a C++.

- C# je navržen pro psaní aplikací jak pro zařízení se sofistikovanými operačními systémy, tak pro zařízení s omezenými možnostmi.
- Přestože by programy psané v C# neměly plýtvat s přiděleným procesorovým časem a pamětí, nemohou se měřit s aplikacemi psanými v C nebo jazyce symbolických adres.

1.3.2 Asynchronní klient-server komunikace pomocí socketů v C#

Síťový socket je definován jako abstraktní reprezentace nějakého komunikačního kanálu. Tento mechanismus byl navržen na počátku 80. let v Berkeley během implementace TCP/IP. I přesto, že jsou sockety odjakživa spojeny s protokolem TCP/IP a internetem, jedná se o velmi obecnou vrstvu, kterou lze beze změn rozhraní použít pro komunikaci v rámci jednoho stroje přes internet protokoly IPv4 nebo IPv6.

V jazyku C# je pro komunikaci pomocí socketů využita třída System.Net.Sockets.Socket. Zmíněná třída poskytuje množinu metod pro synchronní nebo asynchronní komunikaci.

Při použití synchronní komunikace server čeká na data doručená klientem. Jestliže je proud dat prázdný, hlavní vlákno čeká, dokud jsou mu nějaká data klientem doručena. Takový přístup znamená, že server nemůže během čekání dělat nic jiného a nemůže přijímat žádná data od ostatních klientů. Takové chování není v reálném světě příliš akceptovatelné. A to především z důvodů, že obvykle musí server komunikovat s více klienty, nežli jen s jedním.

V asynchronní komunikaci funguje server tak, že zatímco server naslouchá nebo přijímá data od jednoho klienta, stále může obsluhovat ostatní klienty a plnit jejich požadavky. Při využití asynchronního modelu komunikace, separátní vlákno (na úrovni operačního systému) naslouchá na socketu. Jakmile nastane na socketu nějaká událost, zavolá se tzv. callback funkce, jež je specifikována předtím, než je naslouchání na socketu zahájeno. Callback funkce poté odpoví a zpracuje událost, která na socketu nastala.

Asynchronní komunikaci by bylo možné docílit i využitím klasické více vláknové aplikace. Jazyk C# a .NET framework ale umožňuje tuto komunikaci provádět bez nutnosti více vláken pomocí výše zmíněné třídy System.Net.Sockets.Socket.

1.4 Windows Powershell

Windows PowerShell (dříve známý jako Microsoft Shell, MSH či pod kódovým označením Monad) je rozšiřitelný shell se skriptovacím jazykem od společnosti Microsoft. Produkt je založen na platformě .NET Framework a z toho vyplývá i jeho odlišnost od ostatních shellů, místo textové roury jako je tomu u [UNIX] shellu, obsahuje PowerShell rouru objektovou.

Windows Powershell poskytuje všechny možnosti platformy na které je postaven, tudíž vše co je obsaženo v Microsoft .NET Frameworku je dostupné i z Powershellu. Díky této provázanosti poskytuje Powershell velké množství funkcí pro správu pomocí tzn. cmdlets (vyslovuje se commandlets), které jsou specializované třídy .NET implementující určitou operaci. Skripty Powershell (přípona .ps1) jsou kompozicí cmdletů s podporou logických podmínek. Powershell je nástupcem příkazového řádku Windows, tudíž dokáže pracovat s klasickými aplikacemi Windows (net.exe, ping.exe,...), ale také dokáže vytvářet instance libovolné .NET třídy, případně COM objekt [13].

II. PRAKTICKÁ ČÁST

2 NÁVRH SYSTÉMU

2.1 Stručný popis systému

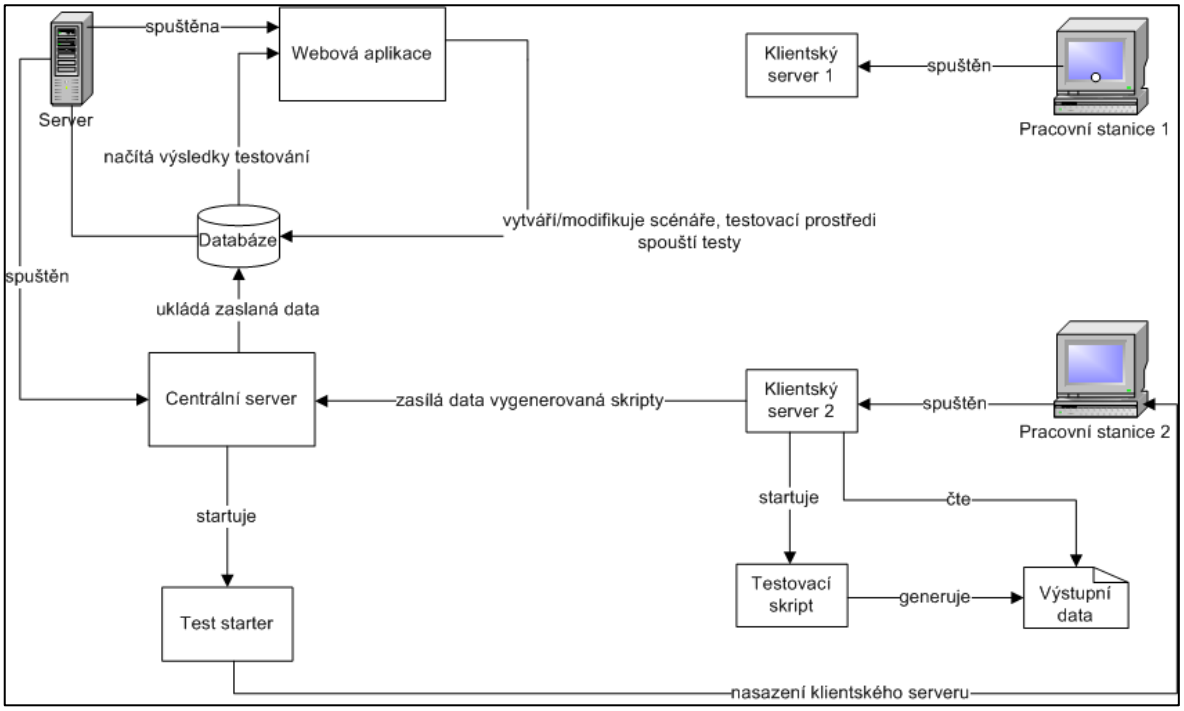
Systém je primárně určen ke spuštění testů, sběru dat a pro následnou vizualizaci dat. Využít jej mohou vývojáři software, kteří si pomocí řešeného systému mohou otestovat novou verzi svých řešení. Systém je navržen tak, aby nesloužil pouze ke sběru benchmarkingových dat. Je možné s jeho pomocí spustit jakýkoli skript nebo program, který určenou aplikaci otestuje a výsledek je zaslán do databáze. Z databáze jsou pak data vizualizovaná pomocí webové aplikace, jenž uživateli umožňuje zobrazit výsledek testu v přehledných tabulkách.

Systém samotný se skládá ze dvou částí.

První část tvoří aplikace klient-server, jenž má na starosti spouštění testů, sběr dat z běžících testů a jejich ukládání do databáze. Aplikace server (dále Master server) sleduje frontu a pokud do fronty přijde nezpracovaný požadavek, začne jej zpracovávat. Zpracování obnáší spuštění aplikace TestStarter, která připraví klientský operační systém tak, aby na něm bylo možné spustit klientský server. Klientský server se po spuštění připojí do Master serveru a začne zpracovávat úkol, jenž mu byl zadán. Pojem úkol znamená v tomto případě skript nebo program, který provádí předem určené akce (instalace programu, spuštění testovaného programu,...) a výsledky těchto akcí zaznamenává v předem definovaném formátu do souboru XML. Klientský server poté už jenom posílá data z XML souboru Master serveru. Master server zaslaná data uloží do databáze. Tímto systémem je tvořeno spuštění testů a sběr dat.

Druhá část se sestává z webové aplikace, přes kterou lze jednoduše definovat prostředí, na kterém test poběží, definovat scénáře, které se budou spouštět a samozřejmě i startovat jednotlivé testy. Dále bude možné skrz systém prohlédnout výsledek testu, posbíraná data a též vizualizovat výsledky např. zátěžových benchmarkingových testů.

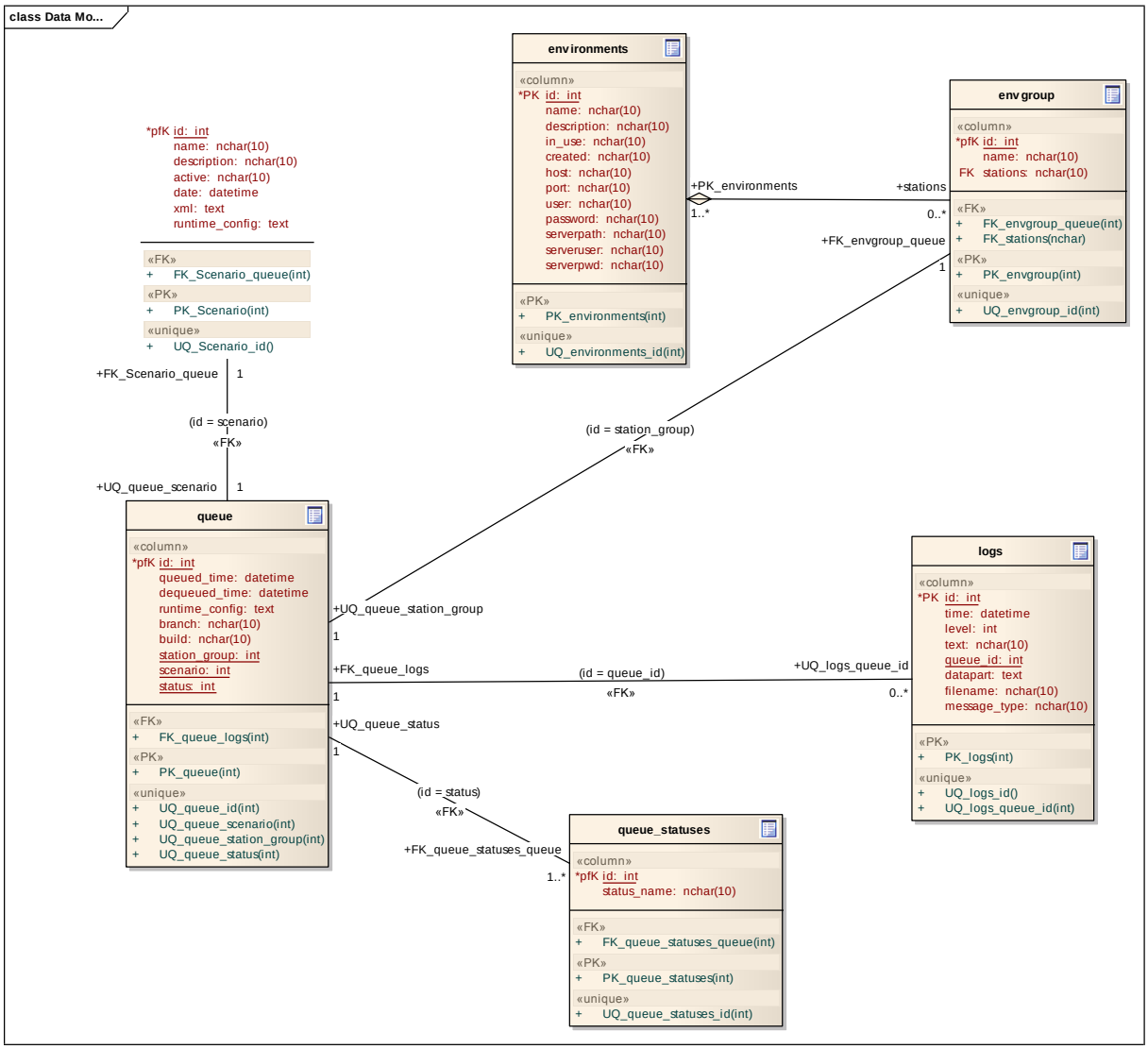
2.2 Diagram fungování systému



Obr. 1. Diagram fungování systému

Na obrázku je názorně popsáno základní fungování systému, kdy na počátku běží pouze centrální server a webová aplikace. Webová aplikace spustí test, který se uloží do fronty. Centrální server na základě nové úlohy ve frontě spouští aplikaci Test Starter, která nasadí na cílovou testovací stanici klientský server včetně testovacích skriptů. Klientský server začne po svém nasazení a spuštění postupně startovat testovací skripty, které budou generovat svá výstupní data v předem dohodnutém formátu. Klientský server bude tyto data číst a posílat zpět centrálnímu serveru, který je uloží do databáze. Pomocí webové aplikace si po zaslání výsledků může uživatel prohlédnout výsledky testu uložené v databázi.

2.3 E-R diagram



Obr. 2. E-R diagram

2.3.1 Popis entit

queue

Fronta, ve které jsou uloženy testy, které doběhly, běží a nebo teprve budou spuštěny

Vytvoření instance: Spuštění nového testu

Modifikace: převzetí testu z frontu nebo dokončení testu

Zrušení: stornování testu, jenž ještě nebyl spuštěn

scenario

Scénář, podle kterého je test spuštěn

Vytvoření instance: zadání nového scénáře do systému

Modifikace: změna scénáře

Zrušení: Smazání scénáře z databáze

environments

Prostředí, na kterém budou testy spuštěny

Vytvoření instance: zadání nového testovacího prostředí do systému

Modifikace: změna testovacího prostředí

Zrušení: smazání nebo zrušení testovacího prostředí

envgroup

Skupina prostředí, na kterém bude test spuštěn

Vytvoření instance: vytvoření nové skupiny testovacích stanic

Modifikace: přidání nebo odebrání stanice ze skupiny

Zrušení: smazání skupiny stanic

logs

Logy, které jsou zapisovány jednotlivými testy

Vytvoření instance: doběhnutí určitého kroku v testu

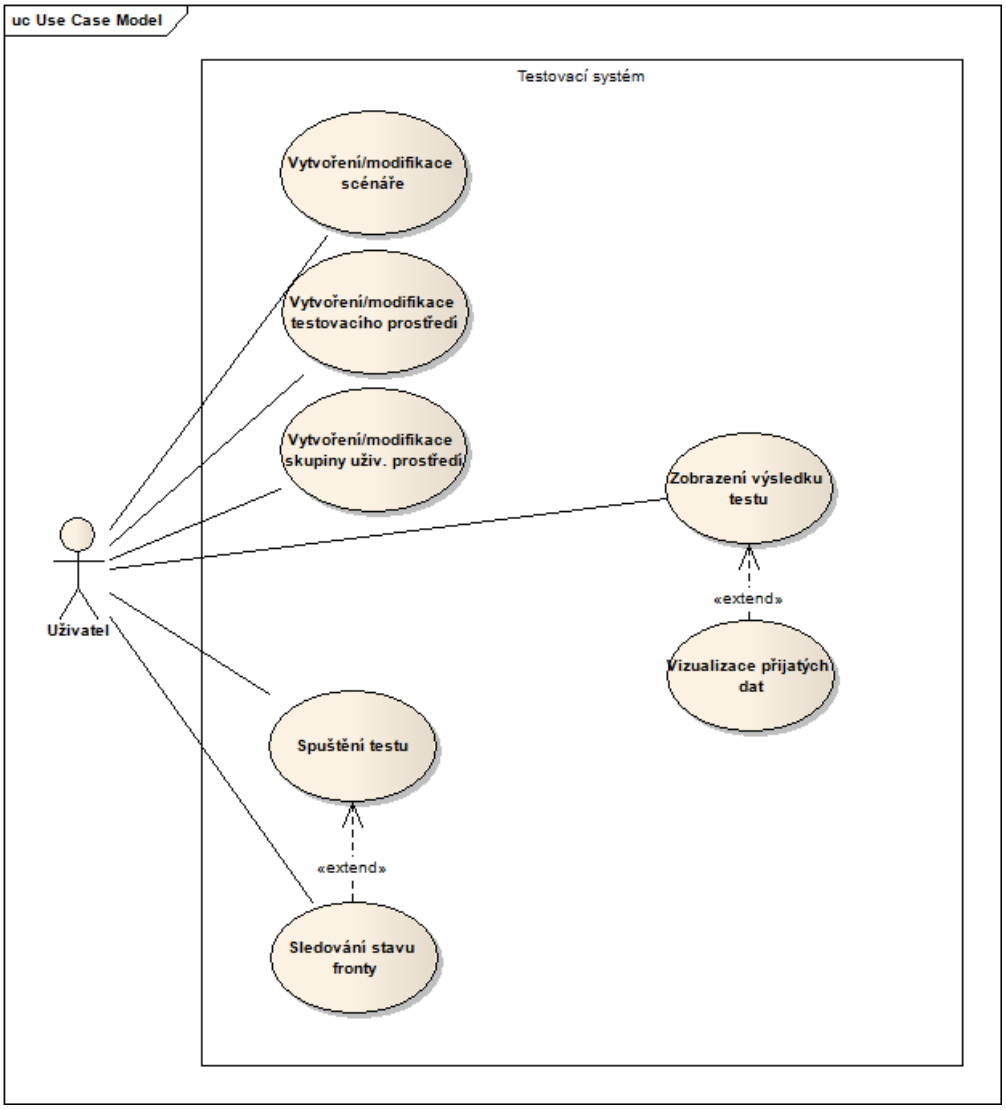
Modifikace: není možná

Zrušení: Smazání výsledku testu z databáze

queue_statuses

Statusy úkolů, které jsou zaneseny ve frontě. Jedná se o pevně danou tabulku v databázi s již předvyplněnými daty.

2.4 Use Case diagram



Obr. 3. Use Case diagram

2.4.1 Kategorie uživatelů

Uživatel

- vytvoří nebo modifikuje testovací scénáře
- vytvoří nebo modifikuje testovací prostředí složené z reálných testovacích strojů
- vytvoří nebo modifikuje skupiny testovacích stanic
- spustí připravený test
- sleduje frontu testů
- zobrazí výsledek testu

3 ŘEŠENÍ APLIKACE

3.1 Klient server aplikace

Jak už bylo zmíněno v předchozích kapitolách, stavebním kamenem aplikace je klient-server aplikace, která zprostředkovává následující akce:

- spouští testy na definovaných testovacích prostředích
- testy jsou spouštěny dle připravených scénářů
- připravuje testovací prostředí na spuštění testů
- centrální server (master server) udržuje komunikaci s klientskými servery
- klient server se stará o spuštěný test a posílá výsledky do centrálního serveru
- centrální server výsledky sbírá a ukládá do databáze

3.1.1 Centrální server

3.1.1.1 Obecně

Konzolová aplikace, která čeká, než se připojí klientský server a poté naslouchá a přijímá posílané data.

3.1.1.2 Konfigurace

Pro konfiguraci serveru slouží XML soubor, jenž se zadává při spuštění služby jako parametr "-configXML".

Konfigurační XML soubor musí nutně obsahovat následující elementy (i s příklady hodnot):

```
<master_server_config>
    <masterIP>192.168.195.247</masterIP>
     <masterPort>8000</masterPort>
     <dbhost>DAVIDR\SQLEXPRESS</dbhost>
     <database>newuss</database>
     <dbuser>sa</dbuser>
     <dbpwd>password</dbpwd>
     <sourceFiles>\\192.168.195.247\c$\newuss</sourceFiles>
     <shared>\\192.168.195.247\c$\server\files</shared>
</master_server_config>
```

Kromě kořene XML nazvaného master_server_config, je obsažen element masterIP, který definuje IP adresu, na němž centrální server bude spuštěn. Druhým elementem masterPort lze definovat port, na kterém bude centrální server naslouchat.

Elementy dbhost, database, dbuser a dbpwd nám definují přístup k databázi. Kde dbhost značí počítač (jméno či IP adresu) na kterém se databáze nachází, database definuje jméno databáze do níž se data ukládají. Elementy dbuser a dbpwd už jsou pouze přístupové údaje k databázi.

Poslední dva elementy jsou přítomny z důvodu sdílení souborů mezi klientskými stanicemi a serverem. Především z důvodu instalace klientských serverů, přenosu logovacích souborů nebo souborů obsahujících data pro tvorbu grafů. sourceFiles definuje cestu ke zdrojovým souborům klientských serverů. Tyto jsou zkopírovány na testovací stanice během instalace klient serverů.

Element shared poté definuje cestu ke sdíleným souborům, kde je možné kopírovat logovací soubory apod. Klientské stanice musí mít do této cesty právo čtení a zápisu. Centrálnímu serveru stačí pouze právo čtení.

3.1.1.3 Třídy, konstruktory a funkce

Centrální server obsahuje základní třídu SocketServer, která obsahuje všechny konstruktory, funkce a proměnné.

SocketServer

Prvním konstruktorem je SocketServer. Konstruktor ve své postatě pouze nahrává konfiguraci z XML souboru pomocí funkce LoadConfiguration a spouští funkci StartListen, pomocí které začne server naslouchat na definovaném portu.

SocketPacket

Zdrojový kód obsahuje též konstruktor SocketPacket, jenž představuje packet posílaný sockety a uchovává klientské číslo připojeného klient serveru a obsahuje také posílané data.

```
1  public class SocketPacket
2  {
3     public SocketPacket(System.Net.Sockets.Socket socket, int
            clientNumber)
4     {
5           m_currentSocket = socket;
6           m_clientNumber = clientNumber;
7     }
8  public System.Net.Sockets.Socket m_currentSocket;
9  public int m_clientNumber;
```

```
10 public byte[] dataBuffer = new byte[1024];
11 }
```

LoadConfiguration

Funkce LoadConfiguration načítá všechny parametry z XML souboru popsaného v kapitole 3.1.1.2. s využitím funkcí z jmenného prostoru System.Xml. Příklad načtení jedné konfigurační položky z XML souboru je pak následující:

```
1 XmlDocument xmlDoc = new XmlDocument();
2 xmlDoc.Load(XMLpath);
3 XmlNodeList masterIP = xmlDoc.GetElementsByTagName("masterIP");
4 IPaddr = masterIP[0].InnerText;
```

V proměnné IPaddr se po provedení příslušného kódu bude nacházet řetězec s IP adresou centrálního serveru.

StartListen

Funkce StartListen vytvoří socket, naváže jej na definovanou IP adresu a port. Jakmile se některý klient připojí, je volána callback funkce OnClientConnect.

```
1 string portStr = port_main;
2 int port = System.Convert.ToInt32(portStr);
3 m_mainSocket = new Socket(AddressFamily.InterNetwork,
  SocketType.Stream, ProtocolType.Tcp);
4 IPEndPoint ipLocal = new IPEndPoint(IPAddress.Any, port);
5 m_mainSocket.Bind(ipLocal);
6 m_mainSocket.Listen(4);
7 m_mainSocket.BeginAccept(new AsyncCallback(OnClientConnect), null);
```

OnClientConnect

Jediným parametrem funkce je objekt typu IAsyncResult, který reprezentuje status asynchronní operace.

```
1 Socket workerSocket = m_mainSocket.EndAccept(asyn);
2 Interlocked.Increment(ref m_clientCount);
3 m_workerSocketList.Add(workerSocket);
4 string msg = "Welcome client " + m_clientCount + "\n";
5 SendMsgToClient(msg, m_clientCount);
6 WaitForData(workerSocket, m_clientCount);
7 m_mainSocket.BeginAccept(new AsyncCallback(OnClientConnect), null);
```

Funkce zkompletuje asynchronní volání BeginAccept() z předchozí funkce StartListen pomocí funkce EndAccept(). Dále inkrementuje počet připojených klientů a pošle klientskému serveru uvítací zprávu. Poslání uvítací zprávy je přítomno spíše z debugovacího hlediska a do budoucna by mělo být zakomponováno do logů. Funkce OnClientConnect dál zavolá funkci WaitForData a uvolní hlavní socket, aby mohli být obslouženi další klienti čekající na připojení.

WaitForData

Vstupem do funkce WaitForData jsou 2 parametry. Jeden typu socket a druhý určuje číslo připojeného klienta. Jakmile je zaznamenána aktivita klienta, je zavolána callback funkce OnDataReceived, jenž přijatá data rozparsuje a uloží do databáze.

```
1  if (pfnWorkerCallBack == null)
2  {
3        pfnWorkerCallBack = new AsyncCallback(OnDataReceived);
4  }

5  SocketPacket theSocPkt = new SocketPacket(soc, clientNumber);
6  soc.BeginReceive(theSocPkt.dataBuffer, 0,
   theSocPkt.dataBuffer.Length,
7  SocketFlags.None, pfnWorkerCallBack, theSocPkt);
```

OnDataReceived

Vstupem funkce je pouze jediný parametr typu IAsyncResult. Funkce spustí zpracování přijatých dat v novém vláknu. A to především z důvodu, že přijatá data mohou být objemnější, pokud by obsahovaly i datovou část a bylo nutné načítat obsah souborů (v případě poslání například logů).

```
1  ReceiveDataThreads oDataReceiveThread = new
   ReceiveDataThreads(asyn);
2  Thread DataReceiveThread = new Thread(new
         ThreadStart(oDataReceiveThread.OnDataReceivedWork));
3  DataReceiveThread.Start();
```

Pro samotné zpracování přijatých dat je volána funkce OnDataReceivedWork.

OnDataReceivedWork

Funkce slouží k přijetí dat a jejich rozparsování. Na úvod je opět dokončeno asynchronní volání BeginReceive(). Poté jsou převedeny přijaté data do proměnné typu string. Pro rozparsování slouží funkce ParseReceivedData. Ta převede přijaté data do pole řetězců. Pole řetězců má předpokládanou strukturu:

["ID úkolu ve frontě, který poslal data", "typ zprávy", "úroveň zprávy", "text zprávy", "cesta ke sdílenému datovému souboru", "jméno původního souboru"]

```
1  int iRx = socketData.m_currentSocket.EndReceive(asyn);
2  char[] chars = new char[iRx + 1];
3  System.Text.Decoder d = System.Text.Encoding.UTF8.GetDecoder();
4  int charLen = d.GetChars(socketData.dataBuffer, 0, iRx, chars, 0);

5  System.String szData = new System.String(chars);

6  List<string> ParsedData = new List<string>();
7  ParsedData = ParseReceivedData(szData);
```

Po rozparsování zprávy jí funkce uloží do databáze. Nejdříve je nutné otestovat, zda zpráva obsahuje datovou část nebo nikoliv. Pokud zpráva obsahuje datovou složku, má vždy více jak 4 složky. Pokud tomu tak je, do databáze je zaznamenáno jméno souboru, na který zpráva odkazovala.

Pro zápis do databáze je použita funkce WriteDatabase, která po dokončení zápisu vrátí ID nově vytvořeného záznamu.

Před zavoláním zápisu do databáze je složen příkazový řetězec INSERT. Ten je utvořen v následující formě:

```
insertstring = "INSERT INTO logs (time, [level], text, queue_id,
      datapart, filename, message_type) " + "VALUES " + "('" +
      datestring + "','" + ParsedData[2] + "','" + ParsedData[3] +
      "','" + ParsedData[0] + "','" + DBNull.Value + "','" +
      DBNull.Value + "','" + ParsedData[1] + "') SET @ID =
      SCOPE_IDENTITY()";
```

Po vytvoření záznamu je do databáze také přenesen obsah souboru, který zpráva obsahovala. To je provedeno v samostatném vláknu, aby se nemuselo čekat na přenesení celého souboru do databáze a mohla být zpracována další zpráva. Přenesení obsahu souboru do databáze je realizováno pomocí funkce LoadSharedFile.

WriteDatabase

Vstupem do funkce pro zapsání záznamu do databáze jsou připojovací údaje k databázi včetně uživatelského účtu. Součástí vstupních parametrů je i již předem připravený příkaz pro vložení do databáze. Pro vložení do databáze jsou použity funkce a metody z jmenného prostoru System.Data.SqlClient.

```
1  SqlConnection MyConnection;
2  SqlCommand MyCommand;
3  int id = 0;


4  MyConnection = new SqlConnection();
5  MyConnection.ConnectionString = @"Initial Catalog=" + database + ";
   Data  Source=" + dbhost + "; Asynchronous Processing=true; User
   ID=" +      dbuser + "; Password=" + dbpwd + ";";


6  MyCommand = new SqlCommand();
7  MyCommand.CommandText = insertstring;
8  MyCommand.CommandType = System.Data.CommandType.Text;
9  MyCommand.Connection = MyConnection;


10 SqlParameter QueueIDParameter = new SqlParameter("@ID",
   SqlDbType.Int);
11 QueueIDParameter.Direction = ParameterDirection.Output;
12 MyCommand.Parameters.Add(QueueIDParameter);
13 if (parameters != null)
14 {
15       foreach (SqlParameter i in parameters)
         {
               MyCommand.Parameters.Add(i);
         }
16 }
17 try
18 {
         MyCommand.Connection.Open();
         MyCommand.ExecuteNonQuery();
         if (newrecord) id = (int)QueueIDParameter.Value;
19 }
```

LoadSharedFile

Funkce LoadSharedFile ukládá soubor, jenž byl připojen ke zprávě do databáze. Funkce pouze aktualizuje již vytvořený záznam v databázi, který je doposud bez obsahu přiloženého souboru. Takto je funkce realizována z důvodu, že přenos většího souboru do databáze neblokuje zpracovávání další zprávy.

```
1  List<SqlParameter> tempparameters = new List<SqlParameter>();
2  byte[] fileData = ReadFile(SharedFileName);

3  System.Text.Decoder d = System.Text.Encoding.UTF8.GetDecoder();
4  char[] asciiChars = new char[d.GetCharCount(fileData, 0,
   fileData.Length)];
5  d.GetChars(fileData, 0, fileData.Length, asciiChars, 0);
6  string asciiString = new string(asciiChars);

7  string insertstring = "UPDATE logs SET datapart=@FileContent WHERE
   ID=" + logID;

8  SqlParameter param = new SqlParameter();
9  param.ParameterName = "@FileContent";
10 param.Value = asciiString;

11 tempparameters.Add(param);

12 WriteDatabase(dbhost, database, dbuser, dbpwd, insertstring,
   tempparameters,false);
```

Na konci je opět spuštěna funkce WriteDatabase, jenž zapíše získaný obsah souboru do databáze.

3.1.2 Klientský server

3.1.2.1 Obecně

Jedná se o konzolovou aplikaci, která spouští testovací skripty na stroji, kde je nainstalována. Sleduje výstupní soubor skriptu a posílá data v předem daném formátu do centrálního serveru, kde se ukládají do databáze.

3.1.2.2 Konfigurace

Pro konfiguraci serveru slouží XML soubor, jenž se zadává stejně jako pro centrální server při spuštění služby jako parametr "-configXML". Příklad konfiguračního souboru klientského serveru:

```
<slave_server_config>
      <masterIP>192.168.195.247</masterIP>
      <masterPort>8000</masterPort>
      <messageFile>.\messages2.msg</messageFile>
      <shared>\\192.168.195.247\c$\server\files</shared>
      <queue_ID>35</queue_ID>
      <workDir>c:\server</workDir>
      <slavejobs>
            <job>
                  <name>install</name>
                  <script>powershell -Command "exit
                        c:\server\scriptxml1.ps1"</script>
                  <finished>0</finished>
            </job>
      </slavejobs>
</slave_server_config>
```

Elementy masterIP a masterPort definují centrální server, do kterého se má klientský server hlásit a kam má posílat data.

Položka messageFile značí soubor, do kterého se ukládají veškerá hlášení posílané skriptem. Shared element ukazuje na složku, do které je možné ukládat soubory, které chceme poslat k uložení do databáze. Položka queue_ID značí identifikační číslo fronty, ze které byla úloha spuštěna. workDir pak značí umístění klientského serveru na disku testovacího počítače. Pro deployment je brána právě tato položka jako složka, kam se mají kopírovat binární a konfigurační soubory klientských serverů.

Konfigurační XML obsahuje sekci slavejobs, kde jsou poté uvedeny všechny úkoly, které má klientský server provést. Obvykle se zde uvádějí skripty, které se mají spustit, aby provedly testovací sekvenci. Element ale může obsahovat i klasické MS DOS příkazy, které je schopen systém provést z příkazové řádky.

Každá sekce, jenž definuje jeden úkol je uvedena rodičovským elementem job. Poté následuje podřízená položka name, která obsahuje jméno úkolu. Element script obsahuje

příkaz, který má být spuštěn. Poslední element finished nemá přímo v konfiguračním souboru žádný význam. Hraje roli až v samostatném seznamu úkolů, kde je po dokončení úkolu označen úkol jako dokončený.

Klientský server taktéž po spuštění vytvoří další XML soubor, kam se ukládají zprávy ze skriptů. Tyto zprávy mají následující formát:

```
<message>
      <type>2</type>
      <level>2</level>
      <text>Toto je text a je k nemu prilozen soubor</text>
      <data>1</data>
      <datapart>C:\Users.starter.py</datapart>
      <originalname>C:\server\starter.py</originalname>
      <sent>1</sent>
</message>
```

Element type určuje o jaký typ zprávy se jedná. Zprávy mohou mít několik typů, které jsou definovány čísly.

- hlášení pro debug text skriptů značí číslo 1
- informaci značí číslo 2
- varování má číslo 3
- chyba je číslo 4
- start sekce značí číslo 5
- konec sekce má číslo 6

Element level je především hodnota pro uživatelské rozhraní. Pokud je poslána zpráva například z nějaké podřízené akce instalaci aplikace, bude mít vyšší level. Tím pádem bude více odsazená, aby bylo poznat, že se jedná o zprávu podřízené akce.

Element text je pouhý text zprávy

Element data značí, zda je přiložena ke zprávě také datová složka

Element datapart ukazuje na soubor, jenž je přiložen

Element originalname uchovává jméno přiloženého souboru

Element sent je příznak určující, zda zpráva již byla odeslána nebo nebyla

3.1.2.3 Třídy, konstruktory a funkce

Zdrojový kód klientského serveru obsahuje základní třídu SocketClient, která obsahuje všechny konstruktory, funkce a proměnné.

SocketClient

Hlavní konstruktor, který nejprve načte konfiguraci klientského serveru. Poté vytvoří soubor, do kterého se zapisují výstupy ze spuštěných skriptů v definovaném formátu.

Konstruktor spouští dvě samostatné vlákna. První vlákno nazvané "MessageWatcher" má na starosti sledování výstupního souboru testovacích skriptů. Jakmile je zapsána nová zpráva, je předána k odeslání do centrálního serveru. Funkce, která má toto na starosti se jmenuje "ConnectAndRun".

Druhé vlákno nazvané "JobStarter" má na starosti spouštění úkolů, jenž jsou definovány v konfiguračním souboru klientského serveru. V rámci vlákna to má na starosti funkce "Run".

```
1  OutputWatcher oOutputWatcher = new OutputWatcher();
2  JobRunner oJobRunner = new JobRunner();

3  InitializeComponent();
4  LoadConfiguration(configXML);
5  System.IO.Directory.SetCurrentDirectory(workDir);
6  CreateSendFile(sendfile);

7  Thread MessageWatcher = new Thread(new
       ThreadStart(oOutputWatcher.ConnectAndRun));
8  MessageWatcher.Start();

9  Thread JobStarter = new Thread(new ThreadStart(oJobRunner.Run));
10 JobStarter.Start();
```

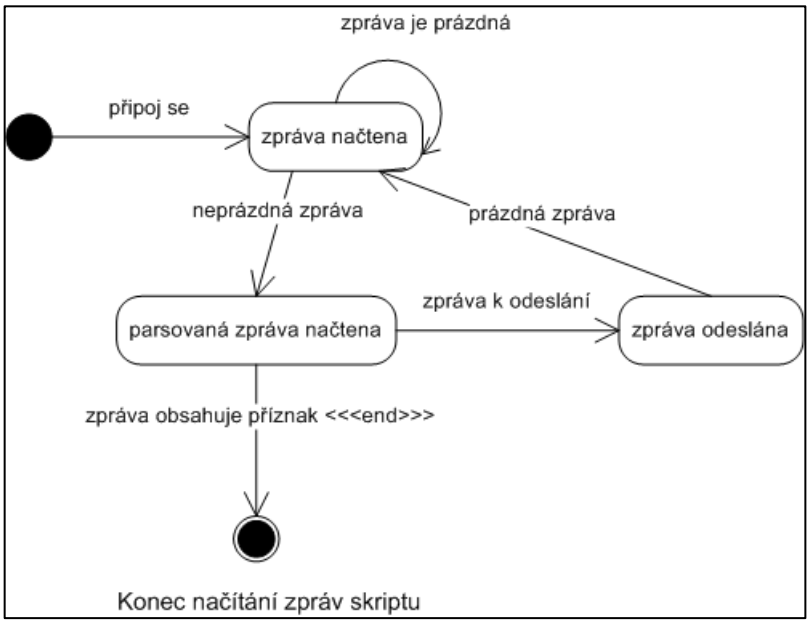
OutputWatcher

Třída, která má na starosti dohled nad souborem s výstupy ze spouštěných skriptů a příkazů. Jak bylo zmíněno, vlákno začíná spuštěním funkce ConnectAndRun().

ConnectAndRun

Na úvod funkce je zavolána jiná funkce z třídy Connect zajišťující spojení s centrálním serverem. Poté probíhá sledování výstupního souboru skriptů pomocí funkce ReturnFileChange. Sledování probíhá ve smyčce, dokud není doručena zpráva "<<<end>>>".

Jakmile je detekována neodeslaná zpráva ve výstupním souboru, zavolá se funkce SendMessage, která odešle zprávu na centrální server.



Obr. 4. Stavový diagram načtení a odeslání zprávy

ReturnFileChange

Funkce, která vrací první doposud neodeslanou zprávu, jenž je uvedena ve výstupním souboru pro skripty. Zpráva je vrácena ve formátu, který je předem definovaný a byl zmíněn v popisu funkce centrálního serveru OnDataReceivedWork. Funkce taktéž kontroluje, zda je ke zprávě připojena datová část či nikoli. V případě, že ano, poskládá zprávu pro centrální server a označí jí jako odeslanou následujícím způsobem:

```
1   output = "<<<" + lstmsgs[i]["type"].InnerText + ">>>,";
2   output += "<<<" + lstmsgs[i]["level"].InnerText + ">>>,";
3   output += "<<<" + lstmsgs[i]["text"].InnerText + ">>>";
4   output += ",<<<" + lstmsgs[i]["datapart"].InnerText + ">>>";
5   output += ",<<<" + lstmsgs[i]["originalname"].InnerText + ">>>";
6   lstmsgs[i]["sent"].InnerText = "1";
```

```
7   xmlDoc.Save(strFilename);
```

Po provedení tohoto kódu se zpráva pro centrální server dostane do tohoto formátu:

```
<<<typ zprávy>>>,<<<level zprávy>>>,<<<text zprávy>>>,<<<datová
část>>>,<<<původní jméno souboru>>>
```

Aby nedocházelo ke konfliktům při využívání sdíleného souboru pro zprávy mezi skripty a klientským serverem, je vždy před přístupem do souboru vytvořen další soubor, který značí uzamčení souboru se zprávami. Soubor uzamyká a odemyká funkce TrafficLightLock respektive TrafficLightRelease.

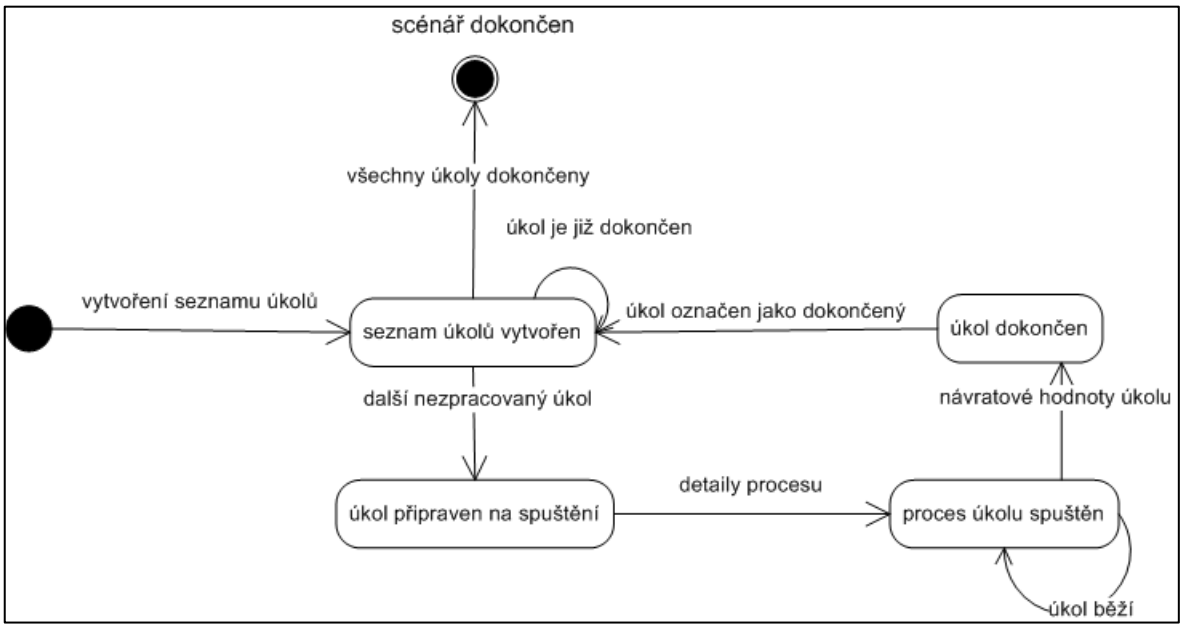
JobRunner

Třída, která obsahuje funkce pro spuštění úkolů definovaných v konfiguračním souboru klientského serveru. Funkce zabývající se samotným spuštěním je funkce Run.

Run

Nejprve jsou úkoly z konfiguračního souboru klientského serveru separovány do samostatného souboru, aby bylo možné do něj libovolně zapisovat. To má na starosti funkce JobsXmlCreate.

Po vytvoření samostatného souboru s úkoly jsou úkoly postupně zpracovávány a označovány jako dokončené. Vyhledávání dosud nesplněných úkolu provádí funkce NextUnfinishedJob a samotnou exekuci úkolů provádí funkce ExecuteCommand.



Obr. 5. Stavový diagram pro spuštění scénáře

ExecuteCommand

Funkce pro spuštění skriptů a příkazů uvedených v konfiguračním souboru klientského serveru. Využívá původních funkcí z jmenného prostoru System.Diagnostics. Každý příkaz je spouštěn přes příkazový řádek "cmd.exe" s parametrem /C, aby došlo k uzavření okna po dokončení skriptu. Funkce vrací návratový kód skriptu.

```
1   ProcessStartInfo ProcessInfo;
2   Process Process;

3   ProcessInfo = new ProcessStartInfo("cmd.exe", "/C " + Command);
4   ProcessInfo.CreateNoWindow = false;
5   ProcessInfo.UseShellExecute = true;
6   Process = Process.Start(ProcessInfo);
7   if (Timeout == 0)
8   {
9         Process.WaitForExit();
10  }
11  else
12  {
13        Process.WaitForExit(Timeout);
14  }
15  ExitCode = Process.ExitCode;
16  Process.Close();
```

3.2 Test Starter

3.2.1.1 Obecně

Test starter slouží pro spuštění vybraného scénáře na konkrétním prostředí. Samostatné spouštění testů se hodí především pro plně automatizované testování, kdy je možné naplánovat spuštění například do plánovače Windows.

Aby bylo možné definovat, který test se má spustit, podporuje Test starter několik parametrů:

- configXML – v parametru musí být definována cesta ke konfiguračnímu XML souboru centrálního serveru
- runtime – parametr definuje cestu k runtime configu uloženému v souboru
- scenario – akceptuje identifikátor scénáře, který má být spuštěn

- env – akceptuje identifikátor skupiny testovacích prostředí, na kterých bude test spuštěn
- build – určuje číslo buildu aplikace, který bude testován
- branch – definuje větev aplikace, která bude testována

3.2.1.2 Třídy, konstruktory a funkce

TaskStarter

Hlavní konstruktor, který řídí celé spuštění testů. Nejprve načte konfiguraci centrálního serveru totožným způsobem jako centrální server. Poté načte pomocí funkce ReadStationGroup jednotlivé stanice ve skupině, na které proběhne testování. Veškeré údaje stanic jsou ukládány do pole stationfordeployment, kde je každý prvek definován třídou station.

Po načtení stanic je z databáze zjištěn scénář testu pomocí funkce ReadScenario a všechny potřebné údaje jsou uloženy do fronty ke zpracování pomocí funkce WriteDatabase.

Následné nasazení klientských serverů je řešeno pomocí funkce DeploySlave pro všechny stanice ve skupině.

ReadStationGroup

Vstupní hodnoty funkce jsou přístupové údaje k databázi a identifikátor skupiny stanic. Z databáze si nejprve funkce načte všechny stanice v definované skupině stanic. Poté postupně získává detaily o jednotlivých stanicích pomocí funkce ReadSingleStation. Ty poté uloží do pole stanic a pole vrací jako návratovou hodnotu.

Jak už bylo zmíněno, funkce ReadSingleStation pouze načítá údaje o stanicích ve skupině a údaje vrací v proměnné typu station.

DeploySlave

Vstupní údaje jsou proměnná typu station, identifikátor úlohy ve frontě a scénář úlohy ve formátu XML. Jelikož funkce kopíruje binární a konfigurační soubory klientského serveru na testovací počítač, je nutné, aby nejdříve získala do stanice přístup. Jsou využity funkce z jmenného prostoru System.Security.Principal, kde se neprve vytvoří nová identita a pomocí metody Impersonate, se pod novou identitou uživatele kopírují soubory a mění registry na cílové stanici, tak aby se po restartu stanice server opět spustil. Nakonec se stanice v této funkci i sama restartuje, aby byl po restartu server automaticky spuštěn.

```
1   IntPtr token = IntPtr.Zero;
2   IntPtr dupToken = IntPtr.Zero;
3   bool isSuccess = LogonUser(computer.user, computer.host,
    computer.password, LOGON32_LOGON_NEW_CREDENTIALS,
    LOGON32_PROVIDER_DEFAULT, ref token);
4   WindowsIdentity newIdentity = new WindowsIdentity( token );
5   using (newIdentity.Impersonate())
6   {
7   try
8   {
9         System.IO.Directory.CreateDirectory(UNCpath);
10        TextWriter slconfig = new StreamWriter(Path.Combine(UNCpath,
          "conf_slaveserver.xml"));
11        slconfig.WriteLine(SlaveConfig);
12        slconfig.Close();
13        RecursiveCopy(sourceFiles, UNCpath);
14        SetRegKey(computer.host,
          @"SOFTWARE\Microsoft\Windows\CurrentVersion\Run",
          "SlaveServer", Path.Combine(computer.serverpath,
          @"NewClient.exe") + " " + "-configXML " +
          Path.Combine(computer.serverpath, "conf_slaveserver.xml"));
15        RestartStation(computer);
16  }
```

3.3 Uživatelské rozhraní

Uživatelské rozhraní je vytvořeno pomocí technologie ASP.NET v Microsoft Visual Studio 2008. Účel uživatelského rozhraní lze shrnout do bodů:

- lze snadno vytvářet a definovat nové testovací prostředí, na kterých budou skripty spouštěny
- testovací prostředí lze sdružovat do skupin, kde je možné spustit jeden testovací úkol
- je možné vytvářet scénáře, které mohou obsahovat libovolné množství úkolů, jenž budou postupně prováděny na testovacích prostředích
- do scénářů lze přidat předdefinované "runtime config" položky. Jedná se o vstupní údaje pro testy, které mohou být modifikovány při každém spuštění
- v sekci pro spuštění testů je možné spustit jak nový test, tak prohlédnout stav fronty
- v manažeru výsledků testů lze prohlížet všechny výsledky proběhlých testů včetně zobrazení datových souborů

Celé uživatelské rozhraní má jednu "master" stránku, která určuje vzhled všech dalších oken rozhraní. Tím je určen vzhled všech dalších stránek uživatelského rozhraní. Master stránka obsahuje 2 komponenty ContentPlaceHolder, do kterých jsou dosazeny příslušné komponenty z jednotlivých stránek.

Uživatelské rozhraní obsahuje 3 základní sekce. Jsou jimi manažer konfigurace, manažer testů a manažer výsledků.

3.3.1 Manažer konfigurace

Přehledy

Manažer konfigurací umožňuje zobrazit všechny dostupné scénáře, testovací prostředí a jejich skupiny. Všechny tyto přehledy jsou uvedeny v tabulce GridView.

Veškerá data pro tabulky jsou získávána z databáze přes komponentu SQLDataSource.

Jako první sloupec je vždy uvedeno políčko pro editaci jednotlivých záznamů. Buňka editace je vždy typu „HyperLinkField“. Po kliknutí na odkaz „Edit“ je uživatel přesměrován na editační stránku s parametrem ID.

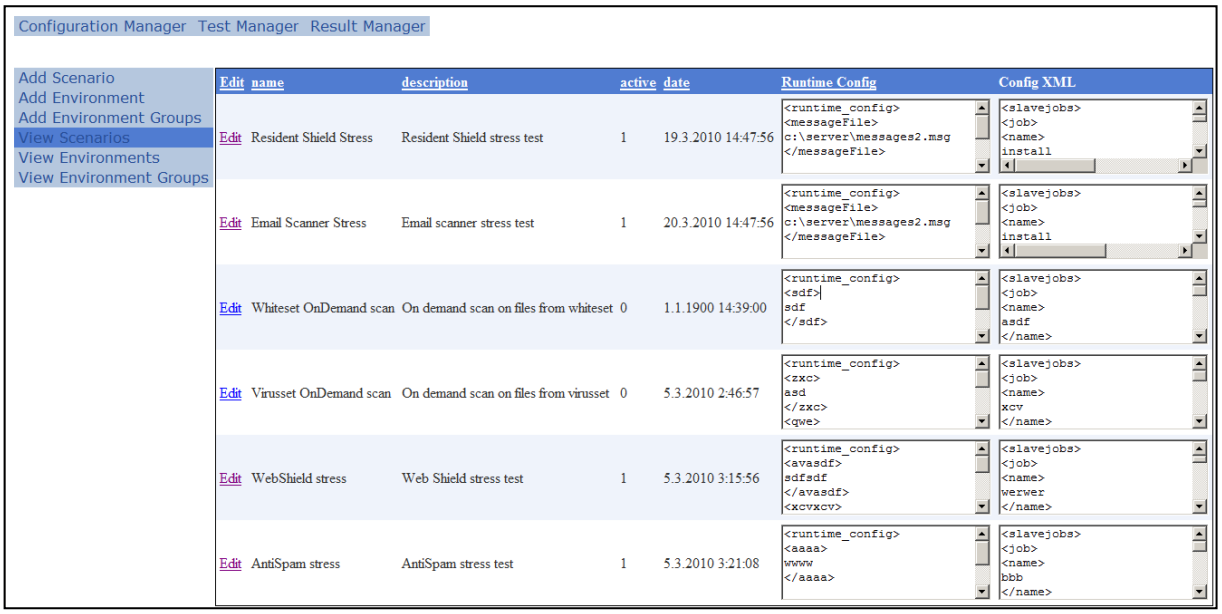
Pro některé typy dat, jenž jsou uloženy v databázi je použita speciální nadefinovaná buňka TemplateField. Do buňky TemplateField je poté možné vložit libovolné komponenty uživatelského prostředí jako tlačítko nebo textové pole.

Například u zobrazení seznamu scénářů jsou data před vložením do tabulky GridView modifikovány, aby byly pro uživatele lépe čitelné. V code-behind souboru se pak nachází funkce, která data modifikuje. U zobrazení scénářů se funkce jmenuje ConvertXML a upravuje zobrazení XML, které je uloženo v databázi jako text na jednom řádku.

Pro přiřazení hodnoty k TextBoxu, jenž je součástí TemplateField se používá v ASP.NET následující syntaxe:

```
<asp:TextBox ID="TextBox1" Text='<%# ConvertXML(Eval("xml"))
%>'</asp:TextBox>
```

V pravé části přehledů je již zobrazeno pouze navigační menu pomocí komponenty Menu.



Obr. 6. Zobrazení seznamu scénářů

Editace a vytvoření nových položek

Manažer konfigurací nabízí také vytvoření a editaci všech položek, které jsou v databázi uloženy. Tj. scénáře, prostředí a skupiny prostředí.

Scénář

Formulář pro úpravu nebo přidání scénáře obsahuje třídu TableRow, do které se ukládají nebo načítají data z databáze.

```
1   public class TableRow
2   {
3   string _id;
4   string _name;
5   string _description;
6   string _active;
7   string _date;
8   string _xml;
9   string _runtime_config;

10  public string ID
11  {
          get { return _id; }
          set { _id = value; }
12  }
```

```
13   public string name
14   {
           get { return _name; }
           set { _name = value; }
15   }
16   public string description
17   {
           get { return _description; }
           set { _description = value; }
18   }
19   public string active
20   {
           get { return _active; }
           set { _active = value; }
21   }
22   public string date
23   {
           get { return _date; }
           set { _date = value; }
24   }
25   public string xml
26   {
           get { return _xml; }
           set { _xml = value; }
27   }
28   public string runtime_config
29   {
           get { return _runtime_config; }
           set { _runtime_config = value; }
30   }
31   }
```

Vzhledem k tomu, že scénáře mohou obsahovat libovolný počet položek Runtime Config a úkolů (Jobs), je nutné generovat textové pole pro zadávání jejich hodnot dynamicky. Textové pole jsou generovány do komponent PlaceHolder. V rámci kódu v pozadí jsou ukládány do pole, kde je typem položky TextBox.

Aby byl zachován počet textových polí i při takzvaném PostBacku, který je vyvolán např. stiskem tlačítka „Add Job“, je počet textových polí ukládán pomocí ViewState. U scénářů je do ViewState ukládán počet textových polí úkolů a runtime config položek.

Uložení ViewState proběhne například pomocí kódu:

```
ViewState["tbCount"] = count;
SaveViewState();
```

Načtení posléze:

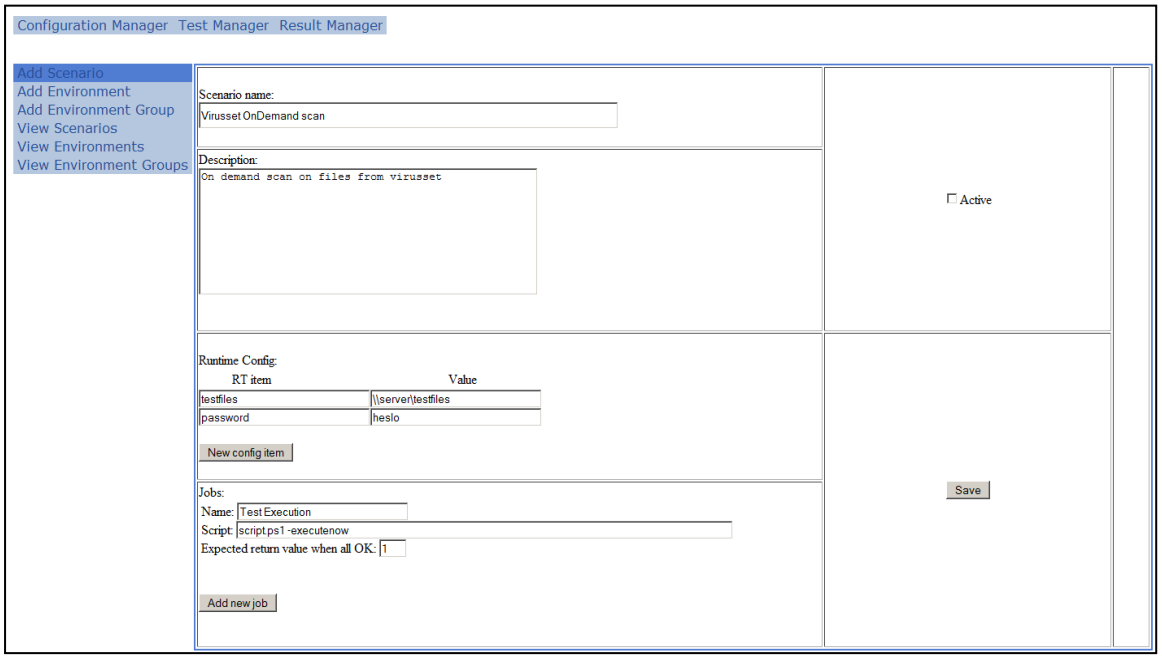
```
Int32.Parse(ViewState["tbCount"].ToString())
```

Pro ukládání hodnot runtime config a úkolů jsou definovány 2 třídy:

```
 1   public class XmlDataRT
 2   {
 3   string _name;
 4   string _value;

 5   public string name
 6   {
           get { return _name; }
           set { _name = value; }
 7   }
 8   public string value
 9   {
           get { return _value; }
           set { _value = value; }
10   }
11   }

12   public class XmlDataJB
13   {
14   string _name;
15   string _script;
16   string _returnvalue;

17   public string name
18   {
           get { return _name; }
           set { _name = value; }
19   }
20   public string script
21   {
           get { return _script; }
           set { _script = value; }
22   }
```

```
23    public string returnvalue
24    {
            get { return _returnvalue; }
            set { _returnvalue = value; }
25    }
26    }
```

XmlDataRT slouží pro uložení dat runtime config a XmlDataJB pro data úkolů. Výsledný design editace scénářů je zobrazen na obrázku 4.5.

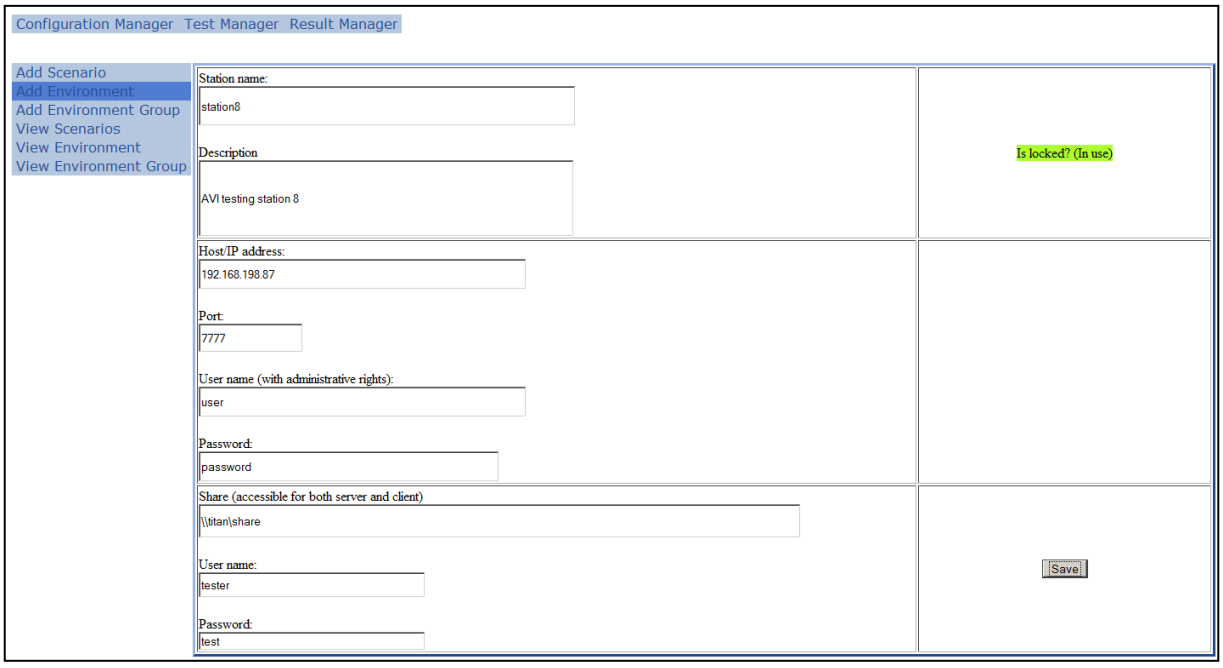


Obr. 7. Editace scénáře

Testovací prostředí

Editace a přidání nových uživatelských prostředí je na rozdíl od scénářů velice jednoduchá. Neobsahuje žádné generované komponenty, a pokud se jedná o editaci již existujícího testovacího prostředí, vše je načteno do proměnné, jenž má uživatelsky definovaný typ TableRow_Env. Z této proměnné je pak vše načteno do textových polí ve formuláři.

Pomocí formuláře je tak možné definovat jméno testovacího prostředí a jeho popis. Poté IP adresu nebo jméno počítače v síti a přístupové údaje k němu. Poslední definovatelnou položkou je sdílený adresář, kam je možné ukládat data a přístup k těmto datům bude zajištěn jak pro lokálního uživatele, tak pro centrální server běžící na jiném stroji.



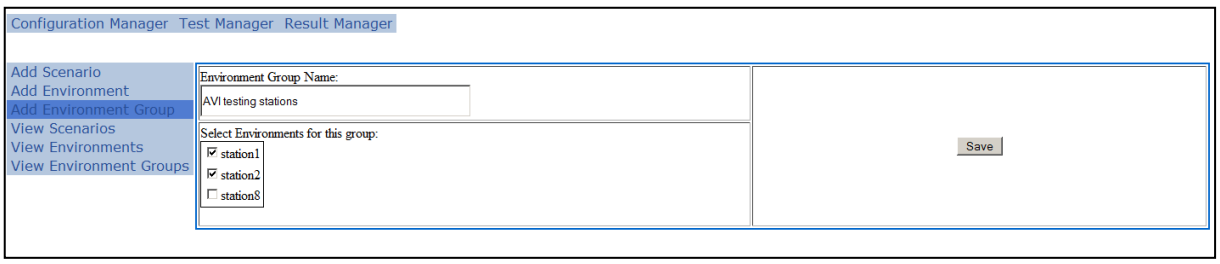
Obr. 8. Přidání nebo editace stanice

Skupina testovacích prostředí

Formulář pro přidání nové skupiny obsahuje generovaný CheckBoxList, ve kterém si může uživatel vybrat, které testovací prostředí bude do skupiny patřit. Zobrazují se pouze aktivní testovací prostředí a jedno uživatelské prostředí může být členem více skupin.

Pokud je spuštěn test na skupině testovacích prostředí a alespoň jedno z nich je ve skupině, na které již běží jiný test, systém před spuštěním vyčká, dokud nejsou všechna uživatelská prostředí uvolněna.

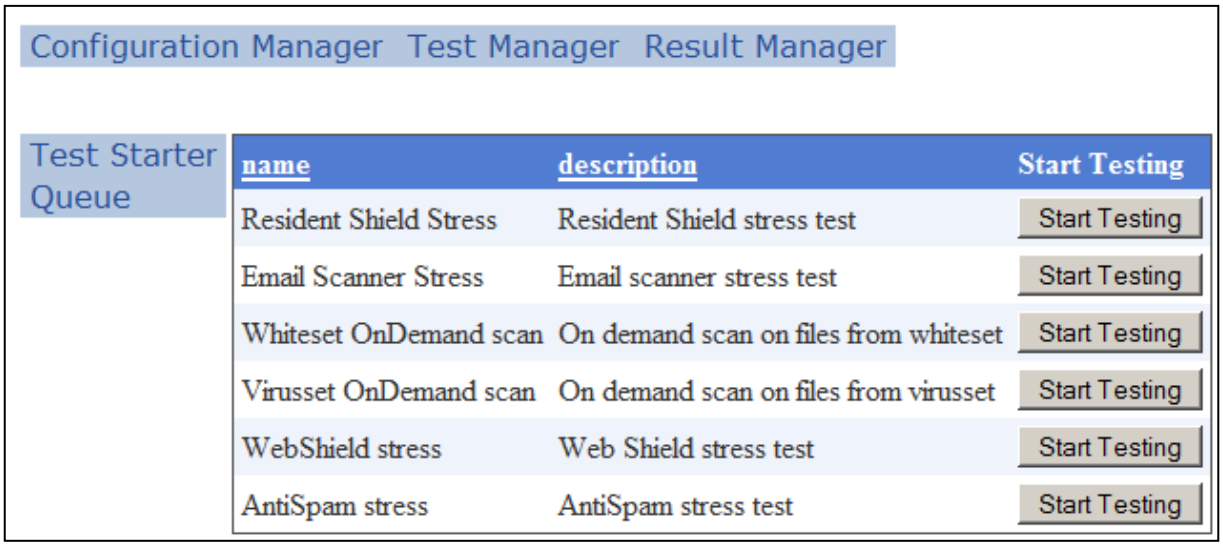
Do formuláře lze nadefinovat ještě jméno skupiny, podle kterého skupinu poznáme. Jméno může být libovolné, jelikož prostředí je v databázi uloženo pod jedinečným identifikátorem.



Obr. 9. Přidání nebo editace skupiny stanic

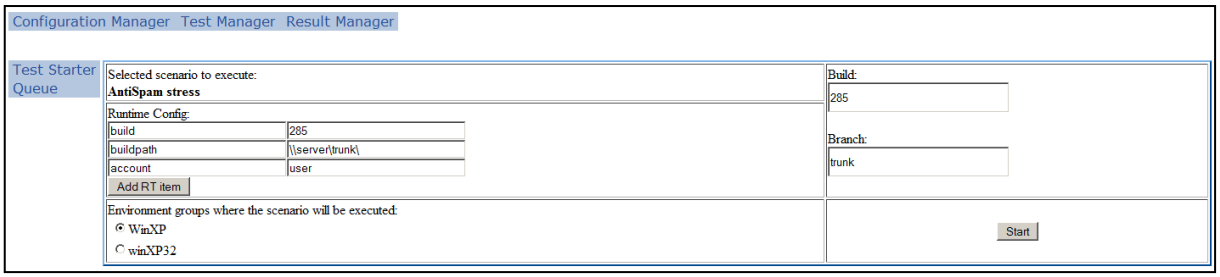
3.3.2 Manažer testů

Manažer testů se skládá ze dvou hlavních částí. První je zaměřen na start testů.



Obr. 10. Výběr testu ke spuštění

Z výběru je možné si zvolit, který test chce uživatel spustit. Po kliknutí na "Start testing" u konkrétního testu se zobrazí detailní volby pro běh testu.



Obr. 11. Zadání detailů pro spuštění testu

Před spuštěním testu je tak možné nadefinovat hodnoty runtime config, které se mohou lišit oproti výchozím hodnotám ve scénáři. Jak už bylo výše zmíněno, runtime config hodnoty jsou uživatelsky definované parametry, které se přenášejí ke spouštěným skriptům. Mohou skriptu poskytovat informace o tom, jaký build testované aplikace se má v rámci testování instalovat nebo odkud má skript stáhnout instalační soubory.

V detailním výběru je možné také zvolit, na které skupině stanic test poběží. Zobrazují se pouze aktivní skupiny stanic.

Pro snazší filtrování ve výsledcích testu je nutné vyplnit položky "build" a "branch", jenž by měly definovat, která verze a větev aplikace byla testována.

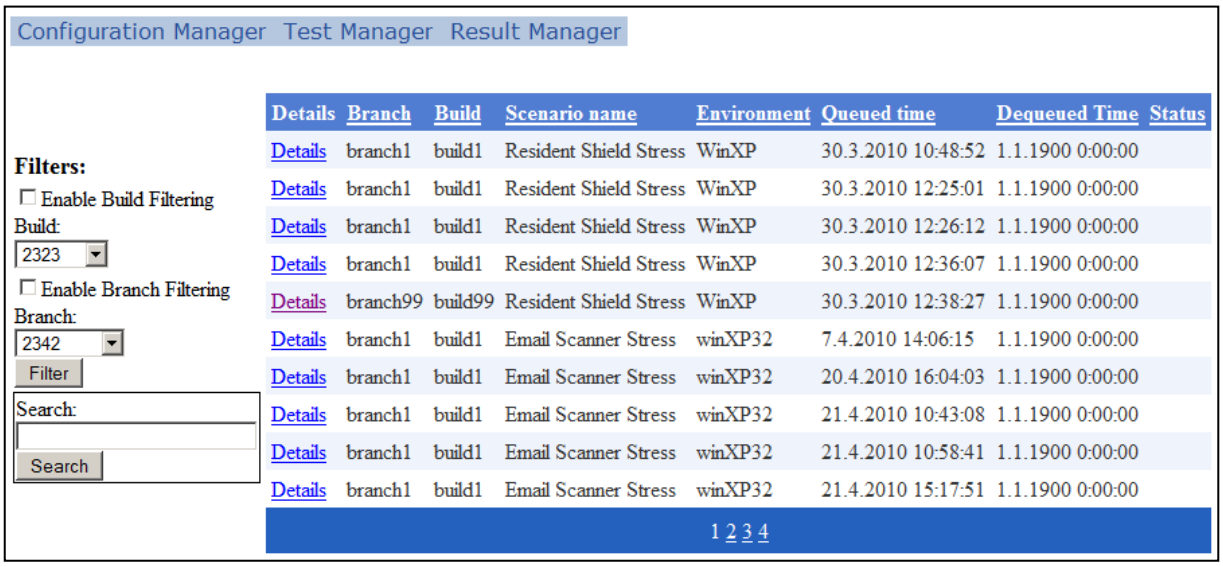
Pro spuštění samotného testu slouží tlačítko Start. Pomocí tlačítka je poté spuštěn TestStarter, který se postará o zařazení do fronty.

Sekce pro prohlížení fronty není zcela doimplementována. Nyní zobrazuje pouze celou frontu bez možnosti jakékoli akce. Doimplementováno by mělo být smazání úkolu z fronty a restart spuštěného úkolu ve frontě.

3.3.3 Manažer výsledků testů

Pomocí sekce manažeru výsledků lze jednoduše procházet všemi výsledky testů. Základní výpis je opět k dispozici v tabulce GridView. Pro snadnější orientaci jsou k dispozici možnosti filtrování. A to podle položky build nebo položky branch. Je taktéž možné ve výsledcích testů vyhledávat. Prohledávány jsou pouze záznamy v databázi, jenž jsou zobrazeny v tabulce. Detailní výsledky testů již prohledávány nejsou. Samozřejmostí je možnost řazení výsledků podle jednotlivých sloupců.

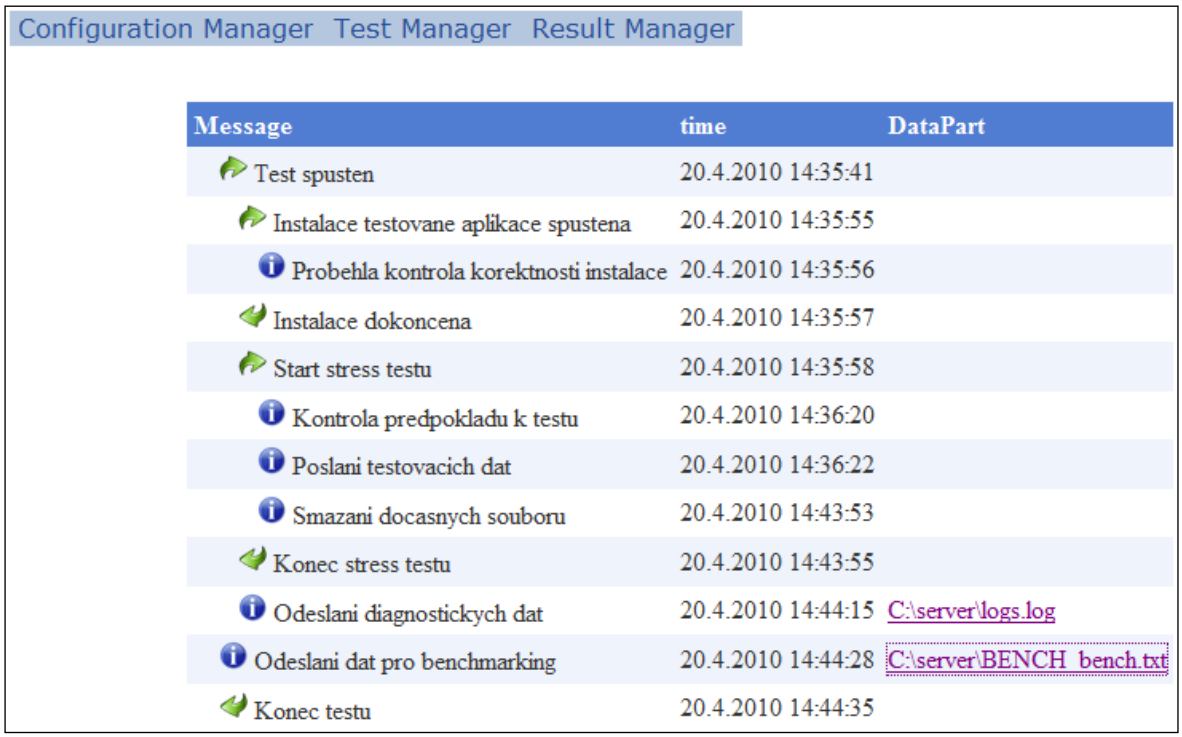
Každý výsledek testu je možné si zobrazit detailněji pomocí odkazu Details u každého výsledku v tabulce.



Obr. 12. Seznam výsledků testů

Zobrazení detailního výsledku testů přesměruje na stránku, kde je možné prohlédnout informace, jenž byly zaslány testem do databáze. Lze tak snadno zjistit, který příkaz byl kterému příkazu podřízen či naopak. U každého záznamu se zobrazuje taktéž ikona, která značí o jaký typ se jedná (informace, start sekce, konec sekce, atd.)

Pokud je přiložen k určitému kroku soubor, je zobrazen ve sloupci DataPart.



Obr. 13. Detail výsledku testu

Po kliknutí na přiložený textový soubor je zobrazen jeho obsah na samostatné stránce. Z té je možné soubor uložit na disk. Odeslání souboru je řešeno funkcí:

```
1   string tempstr = ViewState["filename"].ToString();

2   FileInfo tmpfile = new FileInfo(tempstr);

3   byte[] byteArray = Encoding.ASCII.GetBytes(TextBox1.Text);
4   MemoryStream stream = new MemoryStream(byteArray);

5   Response.ContentType = "text/plain";
6   Response.AppendHeader("Content-Disposition", "attachment;
    filename=" + tmpfile.Name);

7   stream.WriteTo(Response.OutputStream);
8   Response.End();
```

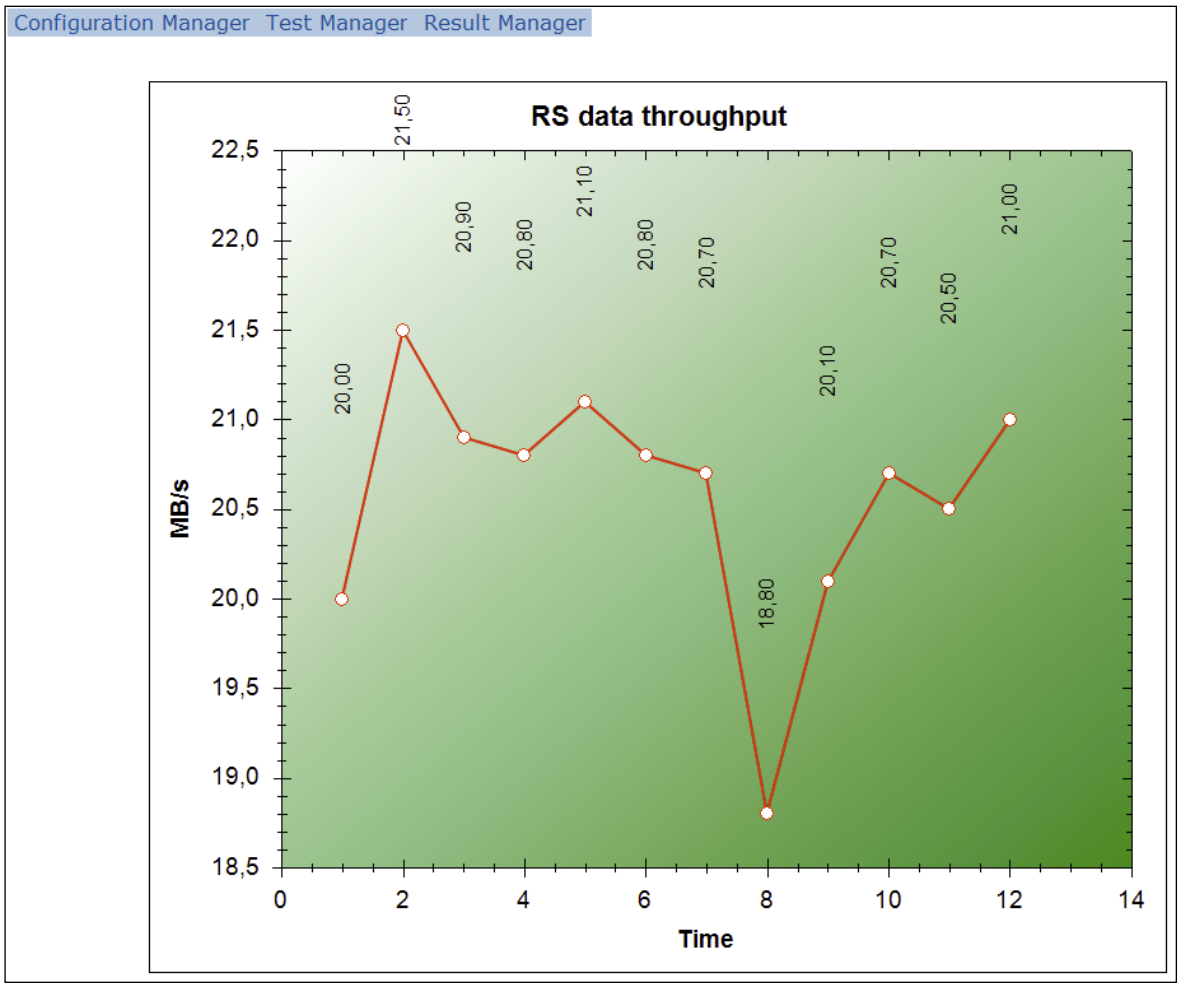
3.3.4 Vizualizace benchmarkingových dat

Do vizualizace je možné se dostat přes kliknutí na datapart v konkrétním výsledku testu. Generování grafického znázornění dat je vytvořeno pomocí knihovny ZedGraph dostupné na internetové adrese: http://zedgraph.org/wiki/index.php?title=Main_Page.

Pro korektní grafické znázornění je data nutné poslat serveru ve formátu:

```
jméno benchmarku;název osy x;název osy y
x1;y1
x2;y2
...
```

Program si převezme textové popisky z prvního řádku souboru a souřadnicemi naplní pole souřadnic PointPairList. Po naplnění pole souřadnicemi se vygeneruje obrázek pomocí knihovny ZedGraph. Prozatím je implementován v systému pouze jeden typ grafu a to standardní spojnicový graf s popisky bodů. Knihovna ovšem nabízí širokou škálu grafů jako sloupcové grafy, pruhové grafy, výsečové grafy nebo bodové.



Obr. 14. Grafické znázornění zaslaného souboru z výsledku testu

3.4 Testovací skripty

3.4.1 Knihovny testovacích skriptů

Aby byly testovací skripty schopny komunikovat s klientským serverem bez problému a posílat mu zprávy v korektním formátu, vytvořil jsem ve skriptovacím jazyku Powershell knihovnu, přes kterou je možné zprávy posílat. Powershell byl využit především díky jeho provázanosti s operačním systémem Windows. Ovšem není problém přepracovat skripty do jiného jazyka např. Pythonu. Systém je taktéž bez problémů spustí a přečte jejich výstupy, pokud budou ve správném formátu.

Pro zasílání zpráv je vytvořena powershell funkce Send-Message. Funkce se volá s parametrem pole, jenž obsahuje čtyři parametry:

- $kwArgs.msg – obsahuje text zprávy
- $kwArgs.type – definuje typ zprávy
- $kwArgs.level – level zprávy
- $kwArgs.file – nepovinný parametr pro případ, že chceme ke zprávě připojit soubor

Dle zaslaných parametrů funkce vygeneruje zprávu pro klientský server ve formátu, jenž je popsán ve článku 4.1.1.2.

Jestliže zpráva obsahuje přiložený soubor, je uložen funkcí na sdílené úložiště definované v konfiguračním souboru klientského serveru. Před samotným uložením je vygenerováno jedinečný GUID identifikátor, který je připojen před jméno souboru. Takto je umožněno, aby na společné úložiště mohly být nakopírovány soubory se stejným názvem. V databázi je poté uchováno původní jméno souboru, takže přes uživatelské rozhraní se soubor k uživateli dostane pod původním názvem.

Aby bylo možné zapisovat do souboru se zprávami pro klientský server a nedocházelo ke konfliktům, kdy stejnou činnost vykonává server, jsou k dispozici dvě funkce LockSendFile a ReleaseSendFile. Funkce LockSendFile soubor zamkne a server tak pozná, že není možné do souboru přistupovat. ReleaseLockFile naopak soubor odemkne. Stejně tak postupuje i klientský server, jak bylo popsáno v předchozí kapitole.

Pro snazší orientaci a zapamatování typů zpráv jsou hodnoty zpráv uloženy do globálních proměnných jako MSG_I pro informaci nebo MSG_E pro chybu.

3.4.2 Testovací skripty

Aby mohly testovací skripty psané v Powershellu využívat naprogramovaných knihoven, nejprve se vždy importují spuštěním daného skriptu s funkcemi. Poté skripty mohou provádět své akce. Jestliže potřebují odeslat klientskému serveru zprávu, zavolají funkci send. Pro odeslání zprávy s přiloženým souborem se volá funkce send s následujícími parametry:

```
send @{msg="Text
zprávy";file="c:\server\log.txt";level=1;type=$global:MSG_I}
```

Jelikož je parametr připojené zprávy nepovinný, zpráva bez přiloženého souboru se volá:

```
send @{msg="Toto je text c. $i";level=$send_level+1;type=$global:MSG_I}
```

3.5 Nedostatky a možné vylepšení aplikace

V této sekci bych rád uvedl některé nedostatky a vylepšení v mnou vytvořeném systému. Jedná se především o funkce, které nebylo možné kvůli jejich rozsahu v rozumném čase implementovat.

3.5.1 Klient - server aplikace

Nevýhodu komunikace mezi klientem a serverem spatřuji především v přenosu souborů pro diagnostiku nebo vizualizaci. Současné řešení využívá sdílení souborů, jehož přednost je menší vytížení sítě při odesílání a příjmu souboru. Lze ovšem předpokládat spoustu problémů v přístupu k souborům. Především co se uživatelských oprávnění týče.

Přenos ostatních zpráv se provádí pomocí socketů. Přes sockety je možné přenést pouze 1kB dat vcelku. Řešením by bylo naslouchat a přijímat data, dokud bude klientský server data posílat a následně data spojit. Bohužel vždy po spojení dat již nebyl soubor binárně shodný s jeho zdrojem. Proto jsem se rozhodl využít řešení sdíleného úložiště, kam má přístup jak klientský server, tak centrální server.

Jiným řešením situace s přenosem souborů by byla implementace webového serveru do aplikace klientského serveru. Soubory, které by se měly přenést do centrálního úložiště, by byly nasdíleny skrze webový server a centrální server by si je stáhl z adresy, jež by mu byla doručena pomocí zprávy přes sockety.

3.5.2 Grafické rozhraní

V grafickém rozhraní shledávám pravděpodobně nejvíce nedostatků. Snažil jsem se jej vytvořit tak, aby především demonstrovalo všechny možnosti, kterých je systém schopen. Tudíž některé funkce v grafickém rozhraní chybí nebo nejsou dotaženy do úplného konce.

V manažeru testů například není možné smazat úkol, který je již zařazen ve frontě. Pro tento případ vyvstává otázka co dělat s úkolem, jenž je právě spuštěn. Aby bylo možné zrušit právě běžící úkol, bylo by nutné modifikovat klient server aplikaci tak, aby přestaly vykonávat zadané úlohy a aby s nimi centrální server přerušil spojení.

Dále by měla být do systému zaimplementována sekce pro komplexní prohlížení výsledků dat pro benchmarking a možnost jejich porovnání například v jednom grafu nebo tabulce. V takové sekci by mělo být možné si vyfiltrovat výsledky testů dle jednotlivých scénářů a uživateli by měla být poskytnuta možnost si zvolit, které výsledky měření by chtěl do grafů zahrnout.

3.5.3 Spolupráce s virtualizačními nástroji

Jako důležité vylepšení systému do budoucna vidím možnost spolupráce s virtualizačními nástroji společnosti VMware jako jsou například VMware Server, VMware vSphere nebo po případě VMware Lab Manager. Do testovacích prostředí by se tak daly nadefinovat stanice, které by v době spuštění testu neběžely. Po spuštění testu by nástroj TestStarter nejprve spustil pomocí VMware API deploy virtuálních stanic. Po dokončení deploye by TestStarter teprve zkopíroval binární soubory klientského server, restartoval stanici a započalo samotné testování.

3.5.4 Notifikace uživatele

Pro uživatele by bylo přínosné, kdyby byl notifikován, jakmile bude konkrétní test dokončen. Vzhledem k tomu, že pomocí systému lze spouštět testy automaticky, například pomocí plánovače úloh Windows, uživatel by mohl po dokončení testu obdržet kupříkladu notifikační email. Takový email by obsahoval čas spuštění a dokončení testu a stručný výsledek testu s počtem korektně dokončených kroků a chyb. V případě chyb by byl k emailu například přiložen kompletní výsledek testu v textové podobě.

3.5.5 Uživatelské role

Systému chybí ke komplexnosti rozdělení uživatelů do alespoň 2 rolí: administrátoři a uživatelé. Administrátoři by měli možnost definovat nová testovací prostředí a scénáře včetně možnosti upravit frontu čekajících testů. Zatímco uživatelé by mohli testy pouze spouštět a prohlížet výsledky testů.

3.5.6 Logování

Současná aplikace vypisuje chyby pouze do konzole v případě klientského nebo centrálního serveru. Webová aplikace se chová velmi podobně a v případě chyby zobrazí hlášení přímo do okna webového prohlížeče. Pro lepší hledání a řešení chyb bych doporučoval zapisovat veškerá chybová hlášení do souboru včetně času, kdy k chybě došlo.

ZÁVĚR

Systém pro sběr dat a následnou vizualizaci dat je funkčním systémem, jímž lze testovat téměř jakoukoli aplikaci na operačním systému Windows, pro kterou je možné napsat testovací skript, jež potřebná data vyprodukuje.

Navrhoval jsem systém s tím cílem, že by neměl být pouze hrubým systémem pro ukládání dat a jejich následnou vizualizaci, ale aby našel i využití pro QA oddělení nejenom AVG Technologies. Tyto požadavky z mého pohledu systém splňuje, i když samozřejmě pro reálné nasazení by bylo žádoucí implementovat a doplnit funkce, které jsem zmínil ve článku 3.4.

Díky spouštění testů přes konzolovou utilitu jej lze použít i jako testovací systém pro automatické testy, které bude možné spouštět bez zásahu člověka. Pokud to umožní testovací prostředí, jako skript, který bude generovat data vhodná pro zanesení do systému, lze použít jakýkoli programovací nebo skriptovací jazyk. Jediný limitující faktor je použití stejného komunikačního protokolu, jaký produkuje knihovna v Powershellu popsaná ve článku 3.3.1.

CONCLUSION

My system for the data collecting and following data visualization is a fully functional system intended for testing almost every application which can be automatically tested by a script on Windows operating system.

I projected the system to achieve a goal, that it should not be a simple system to save the data and its visualization. I also wanted to make it usable for QA departments in (not only) AVG Technologies. From my point of view, my work achieved this goal. Of course, there are some improvements needed to be implemented before it will be used for real work. I mentioned these improvements in the topic 3.4.

Thanks to the console starting system, it is possible to use it as a testing system for automatic tests without any human intervention. It is possible to use any programming or scripting language to generate proper testing data, if testing environment allows it. The only limiting factor is proper communication protocol, which is described in the topic 3.3.1.

SEZNAM POUŽITÉ LITERATURY

[1] Wikipedia [online]. 2010 [cit. 2010-05-20].] Benchmark (computing). Dostupné z WWW: <http://en.wikipedia.org/wiki/Computer_benchmark>.

[2] BROWN, A. B. A Decompositional Approach to Computer System Performance Evaluation. Center for Research in Computing Technology, Harvard University, 1997

[3] GRAY, Jim; REUTER, Andreas. Distributed Transaction Processing: Concepts and Techniques. [s.l.] : Morgan Kaufmann, 1993. 1018 s. ISBN 1-55860-190-2.

[4] DONGARRA, Jack J. Performance of Various Computer Using Standard Linear Equations Software [online]. [s.l.] : University of Manchester, 2009 [cit. 2010-05-30]. Dostupné z WWW: <http://www.netlib.org/benchmark/performance.ps>

[5] DONGARRA, Jack J. Computer Benchmarks (Advances in Parallel Computing). North-Holland, 1993. 364 s. ISBN 978-0444815187

[4] GRACE, R. The Benchmark Book. Prentice Hall, 1996. 313 s. ISBN 978-0133418019

[5] DUJMOVIC, J. J., HOWARD, L. A Method for Generating Benchmark Programs, Department of Computer Science, San Francisco State University, 2000.

[6] ZHANG, Xiaolan. Application-Specific Benchmarking [online]. [s.l.], 2001. 113 s. Dizertační práce. Harvard University. Dostupné z WWW: <http://www.eecs.harvard.edu/syrah/application-spec-benchmarking/publications/thesis.pdf>.

[7] KALIBERA, Tomáš. Performance in Software Development Cycle : Regression Benchmarking [online]. [s.l.], 2006. 174 s. Dizertační práce. Karlova universita v Praze. Dostupné z WWW: <http://d3s.mff.cuni.cz/~kalibera/files/dt.pdf>.

[8] FRY, Ben. Visualizing Data. O'Reilly Media, Inc. 2008. ISBN 978-0-596-51455-6

[9] SAAVEDRA-BARRERA, R. H., SMITH, A. J., Analysis of Benchmark Characteristics and Benchmark Performance Prediction. ACM Transactions on Computer Systems, 1996, ISBN 344-384

[10] Wikipedia [online]. 2010 [cit. 2010-05-20]. C Sharp (programming language). Dostupné z WWW: <http://en.wikipedia.org/wiki/C_Sharp_(programming_language)>.

[11] Wikipedia [online]. 2010 [cit. 2010-05-20]. ASP.NET. Dostupné z WWW: <http://en.wikipedia.org/wiki/Asp.net>.

[12] LONGBOTTOM, Roy. Roy Longbottom's PC Benchmark Collection [online]. 2009 [cit. 2010-05-30]. PC Benchmarks. Dostupné z WWW: <http://homepage.virgin.net/roy.longbottom >.

[13] Microsoft Corporation. Windows PowerShell [online]. 2010 [cit. 2010-05-30]. Dostupné z WWW: <http://www.microsoft.com/windowsserver2003/technologies/management/powershell/default.mspx>.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

QA	Quality assurance – zajištění kvality
API	Application Programming Interface – rozhraní pro programování aplikací
XML	Extensible Markup Language – značkovací jazyk vyvinut konsorciem W3C
HTML	HyperText Markup Language – značkovací jazyk pro hypertext
CSV	Comma-separated values – souborový formát pro výměnu tabulkových dat
PDF	Portable Document Format – souborový formát vyvinutý firmou Adobe
PS	programovací jazyk určený ke grafickému popisu tisknutelných dokumentů
PDA	malý kapesní počítač
I/O	input/output – zařízení, které zprostředkovávají kontakt počítače s okolím

SEZNAM OBRÁZKŮ