VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

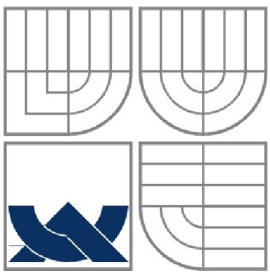
# MONITOROVÁNÍ AKTIVIT UŽIVATELŮ NA PRACOVNÍCH STANICÍCH

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

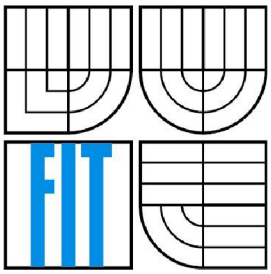
AUTOR PRÁCE
AUTHOR

Bc. Aleš Skopal

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

# MONITOROVÁNÍ AKTIVIT UŽIVATELŮ NA PRACOVNÍCH STANICÍCH

MONITORING OF USER ACTIVITIES ON WORKSTATIONS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Aleš Skopal

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. Petr Matoušek, Ph.D.

BRNO 2009

# Abstrakt

Tato diplomová práce se zabývá návrhem systému pro sledování aktivit uživatelů na pracovních stanicích Windows XP s platformou .NET 2.0. Systém by měl sledovat spuštěné programy, procesy a zjišťovat aktivitu uživatelů v těchto programech, jako je pohyb myší nebo psaní na klávesnici. Součástí celého projektu je návrh systému, do kterého patří specifikace požadavků na takový systém, tvorba architektury systému, tvorba vhodných modelů, návrh komunikace mezi oddělenými částmi systému a zpracování výsledků. Projekt by měl přinést informace nejen o aktivitách uživatelů, ale měl by také poskytnout informace o využití softwaru, který je nainstalován na těchto stanicích. Hotový systém bude otestován formou případové studie. V neposlední řadě se projekt zabývá otázkou etiky v kontextu monitorování uživatelů a okrajově řeší i právní hlediska této problematiky.

# Abstract

This master's thesis deals with proposal of a system for monitoring of user activities on workstations Windows XP over .NET 2.0 platform. The system should monitor running programs, processes and find out user activities in these programs like movement of the mouse or writing on the keyboard. A part of whole project is a suggestion of the system which contains specifications of requirements for this system, creation of the architecture, creation of important models, suggestion of the communication between separated part of system and final compile of results. The project should get information about user's activities and also information about usage of software installed on these workstations. The final system will be tested by use case study. Last but not least, project deals with ethic questions in the context of monitoring of users and discuss law questions about this project.



# Klíčová slova

monitorovací agent, kolektor, sběr dat, personální audit, využití softwaru, etika monitorovaní

# Keywords

monitoring agent, collector, collect of data, personal audit, usage of software, ethic of monitoring


# Citace

Aleš Skopal: Monitorování aktivit uživatelů na pracovních stanicích, **diplomová práce, Brno, FIT VUT v Brně**, 2009

# Monitorování aktivit uživatelů na pracovních stanicích

## Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval zcela samostatně pod vedením Ing. Petra Matouška, Ph. D. a další cenné informace a rady mi poskytl Ing. Roman Ježdík (ALC, spol. s r.o.). Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Aleš Skopal
14.5. 2009

## Poděkování

Tímto bych rád poděkoval Ing. Petrovi Matouškovi, Ph. D. a Ing. Romanovi Ježdíkovi za ochotu, přátelský přístup a věcnou kritiku. Dále bych chtěl poděkovat své přítelkyni Lucii Kyjovské za podporu a pomoc s jazykovou úpravou.

# Obsah

# 1 Úvod

Tento dokument pojednává o tvorbě počítačového systému, jehož hlavním úkolem je sledování aktivit uživatelů a zahrnuje veškerou problematiku s tím spojenou. Věnuje se obecné problematice, která s vývojem systému souvisí a postupně přechází až ke konkrétním návrhům systému, architektuře a nakonec i implementaci.

Důležitým rokem v dějinách informatiky byl rok 1969, kdy byl spuštěn projekt nazvaný ARPAnet. Jednalo se o vytvoření první počítačové sítě. Šlo o propojení čtyř počítačů, které daly vzniknout dnešnímu Internetu. Internet ve svých počátcích (kdy vlastně ještě nešlo o Internet jako takový) sloužil zejména pro vojenské účely. Později se začal rozšiřovat na akademickou půdu a ještě později se stal veřejně dostupným médiem. Velkým krokem v dějinách Internetu se stal projekt www (World Wide Web), který v roce 1991 spustila organizace CERN. Ve chvíli, kdy se začaly vytvářet webové stránky, se začal Internet nezadržitelně rozvíjet. Lidé si začali vytvářet své vlastní lokální sítě, které se mezi sebou spojovaly. To je vlastně celý princip Internetu. Nejedná se o nic jiného, než o síť sítí. Časem se v takové síti začaly objevovat i jiné prvky, než jsou jen stanice a spojení mezi nimi. Jen namátkou se jedná o směrovače (routers), rozbočovače (hubs), přepínače (switches) atp. V dnešní době existují tyto počítačové sítě zejména ve firmách a různých institucích.

Současně se s růstem Internetu pochopitelně rozvíjely aktivity jako je špionáž, šifrování zpráv a kryptografie. Potřeby šifrování zpráv a naopak jejich dešifrování jsou staré jako lidstvo samo. Utajování obsahu zprávy má základy už ve starém Egyptě a kryptografie stojí v pozadí mnoha historických mezníků. S příchodem Internetu nastal problém s tím, že se i obyčejným lidem dostal do rukou nástroj, pomocí kterého mohou komunikovat bez obav, že by se jejich citlivé informace dostaly do nepovolaných rukou. O tématech týkajících se špionáže se zmiňujeme zejména z toho důvodu, že systém na sledování aktivit je jistým způsobem špionážní software.

Cílem projektu je vyvinout systém, který bude schopen monitorovat aktivity uživatele na počítači a zároveň příliš neovlivní jeho soukromí. Důvodů pro vytvoření takového systému je celá řada. V mnoha firmách a podnicích jsou rozsáhlé počítačové sítě, kde neexistuje vhodný způsob, jak ohlídat činnost pracovníků. Dále má jistě každá firma zájem zjistit, zda software, který nakoupila, je skutečně využíván. Chceme proto vytvořit prostředek, který může výrazně zlepšit chod celé firmy. Tedy systém, který bude schopen ohlídat, zda pracovník skutečně vykonává to, co má popř. jakou jinou činností se zabývá, jak hojně je využíván software nainstalovaný na stanicích apod. Existuje jistě řada prostředků, které se o to pokouší, ale většinou se jedná pouze o řešení, která nejsou

komplexně zaměřena na monitorování pracovníků a stanic. Systém, který vyvíjíme by však měl být schopen uspokojit malé, střední i velké podniky.

Pochopitelně již existuje software, který pracuje podobným způsobem jako náš vyvíjený systém. I o něm budeme diskutovat hned v následující kapitole s názvem: „Specifikace požadavků na systém“, kde se zaměříme především na rozdíl mezi těmito existujícími systémy a pokusíme se specifikovat požadavky, které budou kladeny na náš systém. Vytyčíme si hlavní cíle, pro které je systém vytvářen a pokusíme se podrobně popsat všechny činnosti, které musí splňovat. Následovat bude kapitola s názvem: „Etická hlediska a právní limity“, kde odbočíme od technické problematiky, a budeme se věnovat systému z právního a etického pohledu. V této kapitole budeme diskutovat také o možných zneužitích systému a o možnostech, jak zneužití eliminovat. Čtvrtá kapitola nese název: „Modely a architektura systému“. Doplníme si v ní specifikaci požadavků pomocí vhodných modelů a poprvé se setkáme s architekturou vyvíjeného systému. Nebude chybět ani popis celého systému pomocí modelu vrstev, kde se podíváme, jak systém funguje od sledování aktivit, přes přenos dat mezi agentem a kolektorem, uložení dat do databáze až po finální zpracování dat z databáze. Pak následuje kapitola s názvem: „Formát, přenos a bezpečnost dat“, kde se zaměříme na to, jaký je formát ukládaných a přenášených dat. Popíšeme si komunikační protokol, pomocí kterého jsou data přenášena a zohledníme i možné chyby, ke kterým může během přenosu dojít. Pokusíme se rovněž navrhnout zlepšení systému z pohledu bezpečnosti citlivých dat.

V kapitole s názvem: „Implementace systému“ se budeme důkladně věnovat popisu jednotlivých částí systému. Především se zaměříme na nejdůležitější části, ve kterých se pokusíme věnovat pozornost algoritmům. Podrobněji se zaměříme na funkci agenta, vysvětlíme si, jak přesně pracuje, a jak jsou důležité činnosti v systému implementovány. Stejně tak se podíváme na konkrétní úkoly kolektoru a jeho implementaci. Součástí této kapitoly bude rovněž popis klienta pro prezentaci výsledků, kde si povíme o významu většiny souborů, které obsahuje. Vysvětlíme si, proč jsme zvolili právě tuto možnost prezentace dat a budeme se věnovat i technickým detailům, které jsme v této webové aplikaci použili. Nakonec si povíme i o částech systému, které na první pohled nemají tak důležitou úlohu. Předposlední kapitola nesoucí název „Případová studie použitelnosti“ se bude věnovat především praktickému využití systému. Podíváme se, zda je systém snadno použitelný a má dostatek prostředků pro splnění zadání a požadavků.

V závěru dokumentu celou práci podrobně zhodnotíme a podíváme se na přínos systému jako celku. Povíme si, jak by mohl vývoj pokračovat a zda jsme se dostali do fáze, kdy jsme schopni systém nasadit v praxi. Řekneme si rovněž, na co bychom se rádi zaměřili v případném dalším rozvoji projektu.

# 2 Specifikace požadavků na systém

V této kapitole se budeme věnovat především tomu, co od systému vyžadujeme a jaké jsou na něj kladeny požadavky. Vytvoříme tzv. neformální specifikaci systému, zaměříme se na požadavky i na podobné existující systémy. Nejprve si však musíme zavést dva velmi důležité pojmy, které budeme používat v dalším textu. Je nutné určit si role uživatelů systému.

První role bude „*pracovník*“ nebo „*uživatel*“ (budeme používat obě pojmenování), což bude osoba, která pracuje na počítači, který má být sledován. Jedná se o běžného zaměstnance firmy. Druhá role bude „*administrátor*“, což bude osoba, která bude systém spravovat a sledovat výsledky monitoringu. Další důležité pojmy jsou „*agent*“, resp. klient a „*kolektor*“, resp. server. Agent bude ta část systému, která poběží na monitorované stanici a bude získávat data o činnosti pracovníka. Kolektor bude ta část systému, která bude získávat data z agentů a vhodným způsobem je bude ukládat do databáze. Obsah databáze musí být ovšem rozumně prezentován osobě, která bude získaná data kontrolovat, a proto musí celý systém obsahovat intranetový informační systém, který bude prezentovat vysledované informace.

## 2.1 Současný stav monitorovacích systémů

Abychom se mohli zamyslet nad tím, co obnáší vývoj monitorovacího systému, porozhlédneme se po již existujících systémech na monitorování pracovníků. Lze říci, že problematika monitorování je teprve na svém začátku. Proto jsme nalezli pouze tři produkty tohoto typu. První z nich má název „eDetektiv“ a stránky tohoto produktu jsou dostupné z: „http://www.edetektiv.cz/“ [10]. Podle popisu produktu na těchto internetových stránkách se dozvídáme, že „eDetektiv“ pracuje tak, že sleduje nadpisy jednotlivých oken spuštěných aplikací a tím zajišťuje hlídání činnosti pracovníka. Mezi pozitiva tohoto produktu patří zejména to, že neporušuje žádným způsobem zákon, což jak uvidíme u podobných produktů, není vždy takovou samozřejmostí.

Další produkt s názvem „Dozorce“, dostupný z: „http://www.dozorce.cz/“ je na tom o poznání hůř, zejména co se týká souladu se zákony. Sám výrobce na stránkách uvádí: „*Ne vždy a ne ve všech případech je provozování všech druhů monitoringu, a dalších funkcí monitorovacího systému Dozorce, v souladu se zákony* “ [11]. Produkt sleduje i obsahy emailů a podobné informace, které rozhodně nepatří do nepovolaných rukou. Je však o krok dál oproti předchozímu produktu v tom

smyslu, že je schopen získat mnohem více informací. Další výhodou, alespoň podle popisu na stránkách produktu, je zejména inteligentní prezentace dat. Tuto inteligentní prezentaci bychom asi postrádali u třetího produktu tohoto typu, jehož www stránky jsou dostupné z: „http://www.spying.cz/“. Podle popisu na těchto stránkách se zdá, že jsou zobrazovány přímo vysledované informace, jejichž čtení je pro člověka značně náročný úkol [12].

Pokud si shrneme získané informace o těchto produktech, dostaneme několik požadavků, na které budeme klást důraz. Na jejichž základě pak budeme vytvářet i některé modely systému.

## 2.2 Monitorování činnosti uživatelů

Jak jsme si uvedli v úvodu tohoto dokumentu, primárním cílem vyvíjeného systému je monitorování činnosti pracovníků. Výstupem systému by měla být data informující o činnosti pracovníků v podnikové síti na pracovních stanicích. Měl by poskytovat informace o tom, jaké programy pracovník spustil, jak dlouho byly aktivní a jak dlouho v nich byl aktivní, resp. neaktivní, sledovaný pracovník. Aktivitou programu rozumíme dobu, po kterou bylo okno (tedy jeho GUI – graphical user interface) na popředí, anebo dobu, po kterou nedošlo ke změně nadpisu okna (což platí zejména při tzv. surfování po internetových stránkách). Aktivitou pracovníka rozumíme dobu, po kterou se v tomto okně pohybovalo myší nebo se psalo na klávesnici. Ve skutečnosti bude agent zjišťovat dobu, po kterou je uživatel neaktivní, protože z pohledu implementace je to snazší (viz str. 45). Aktivita pracovníka je pak dána jednoduchým vztahem:

$$T_{AP} = F_t(T_{KO}, T_{ZO}) - T_{NP}$$

kde

$F_t$ – funkce, která vyjadřuje rozdíl dvou časů v minutách

$T_{AP}$ – doba, po kterou byl pracovník aktivní [min.]

$T_{KO}$ – čas, kdy přestalo být okno aktivním [časový údaj]

$T_{ZO}$ – čas, kdy se stalo okno aktivním [časový údaj]

$T_{NP}$ – doba, po kterou byl pracovník neaktivní [min.]

Tuto činnost by měl systém hlídat tak, že bude číst nadpisy jednotlivých oken a ukládat je do souboru společně s časovým údajem, kdy se stalo okno aktivní a po ukončení aktivity se zapíše doba neaktivity pracovníka v tomto okně a čas, kdy se stalo okno neaktivním.

Dále musí systém sledovat běžící procesy a informace, které s nimi souvisejí (identifikátor procesu, název, začátek procesu, konec procesu atd.). Pomocí těchto informací jsme pak schopni hlídat i přihlašování a odhlašování uživatelů na stanici a jejich veškerou činnost. Je důležité si uvědomit, že tím, že aplikace bude sledovat pouze nadpisy jednotlivých oken se nedostává do rozporu se zákonem, ve smyslu porušování tajemství dopravovaných zpráv podle znění zákona č. 140/1961 Sb. §239 a §240 v platném znění, částka 65/1961 (viz str. 16-17) [13].

## 2.3 Kontrola využití instalovaného softwaru

Pracovníci potřebují ke své práci určitý software. Často dochází k rozdílným názorům mezi vedením firmy a zaměstnanci na to, který software skutečně ke své práci potřebují. Řešením pro obě strany by měl být námi vyvíjený systém. Měl by podávat informace o tom, které programy byly na stanici spuštěny, jak dlouho byly spuštěny, jak dlouho byly aktivní a jak dlouho se s nimi skutečně pracovalo. Pokud má nadřízený v rukou tato data, poskytují mu informace o tom, který software je skutečně využíván, je potřebný, a který nikoliv.

Díky tomu má dostatečný argument pro případ, že se bude se svými zaměstnanci lišit v názoru na nákup softwaru pro firmu. Principem vyhodnocení využití software bude možnost nechat si vyhledat informace pro daný produkt. Informační systém zodpovědný za prezentaci výsledků bude muset být schopen zpracovat data uložená v databázi a poskytnout detailní informace na základě dotazů, které administrátor vytvoří pomocí různých formulářů. To jistě klade nemalé podmínky na inteligentní řešení prezentace výsledků.

## 2.4 Další vlastnosti systému

Mezi důležité vlastnosti systému by mělo patřit i to, že bude snadno použitelný i pro externí pracovníky využívající přenosné počítače. Je velmi pravděpodobné, že pokud má pracovník možnost využívat firemní notebook i doma, bude jej často využívat i k jiným než pracovním účelům, tj. sledování filmů, hraní her apod. Z toho důvodu nás bude u notebooků pro zaměstnance zajímat především využití software. Zaměstnavatel zřejmě bude počítat s tím, že externí pracovník si rozděluje svůj čas na práci zcela sám. Bude ho ale zajímat, zda využívá software, který mu byl koupen či poskytnut. Tento požadavek má vliv na vývoj systému pouze takový, že nasbíraná data musí být uchována v přenosných počítačích delší dobu, než u běžných stanic, které jsou stále

připojeny k podnikové síti. Dalším požadavkem je, že systém nesmí být klasickou aplikací s uživatelských rozhraním, ale musí se jednat o program, který poběží jako služba. Je to z toho důvodu, že pracovník, který na stanici pracuje, nemusí o systému vůbec vědět a není tak rušen jeho činností.

V neposlední řadě bychom měli zajistit, aby nasbíraná data byla čitelná pouze naším kolektorem. Po otevření tohoto souboru textovým editorem se člověk nedozví prakticky nic, protože budou ukládána v binární podobě. Další výhodou je i to, že soubor bude zabírat méně místa, než kdyby se jednalo o obyčejný textový soubor.

## 2.5 Funkční a nefunkční požadavky

Při návrhu softwaru se často s formulací požadavků velmi dobře vypořádá diagram případů použití. My se s tímto diagramem setkáme v kapitole týkající se modelování systému, do které bez pochyby patří. Místo toho si požadavky, které jsme zjistili při neformální specifikaci, rozdělíme na funkční a nefunkční a k nim přidáme některé nové požadavky. Nejprve bychom si však měli vysvětlit co tyto požadavky znamenají. Funkční požadavky specifikují to, co by měl systém vykonávat. Popisují tedy požadovanou funkci systému. Zpravidla jde o požadavky, kdy systém něco ověřuje, vykonává, kontroluje apod. Nefunkční požadavky jsou takové požadavky, které formulují omezení na systém a vůbec na celý proces vývoje. Jde o podmínky, které omezují i implementaci systému. Nefunkční požadavky se velmi často shodují s požadavky klienta, který si nechává software vytvářet [1].

### 2.5.1 Funkční požadavky

Nyní si specifikujme funkční požadavky na náš systém:

- **Sběr informací o procesech** - systém má na příslušném počítači zjišťovat informace o běžících procesech. Pokud některé procesy nově vzniknout či zaniknout, musí to být vhodně zaznamenáno. Informace, které se budou o procesech zjišťovat, jsou zejména ID (jedinečný identifikátor) procesu, čas, kdy byl proces vytvořen, popř. čas jeho zániku, adresářová cesta k procesu, jméno produktu, ke kterému se proces váže, verze tohoto produktu, verze souboru, ke kterému je proces vázán, jméno společnosti, která produkt vytvořila a originální jméno procesu. Tyto informace jsou získány na základě toho, co vše lze o procesu zjistit a co je pro nás relevantní. V neposlední řadě musíme mít uchovánu informaci o tom, na kterém počítači proces běžel.

- **Sběr informací o aktivních oknech** – systém musí uchovávat informace o spouštění a ukončení aplikací a aktivitě pracovníka v nich. V systému Windows XP je většina programů založena na aplikacích s uživatelským rozhraním. Toto uživatelské rozhraní zpravidla nazýváme okno. Každé takové okno může být v danou chvíli na popředí nebo na pozadí. Nás bude zajímat především chvíle, kdy je toto okno na popředí. V takovém případě budeme často používat termín „aktivní okno“. V každém případě platí, že pokud je příslušné okno neaktivní, je aktivní nějaké jiné, ať už jde o jinou aplikaci, nebo pouze o plochu (desktop), kterou v tomto případě považujeme také za okno. V každém okně musíme sledovat, kdy se stalo aktivním, popř. kdy se stalo neaktivním, nadpis tohoto okna a jak dlouho byl uživatel v tomto okně neaktivní, čímž se myslí doba, po kterou v příslušném okně nepracoval. Musíme sledovat i proces, který toto okno vytvořil, abychom měli dostatek informací např. o výrobci této aplikace, o jejím názvu atp.

- **Informace o stanicích a pracovnících** – informace získané o procesech a aktivních oknech by byly k ničemu, kdybychom nevěděli, ke které stanici jsou tyto informace vázány. Proto je nutné, aby byl systém schopen uchovávat informace o stanicích, na kterých budou nasazeny sledovací agenti. Systém musí znát jméno stanice, aby byl schopen ji „oslovit“ kvůli stažení vysledovaných dat. A pokud už budeme znát jméno stanice, není důvod, proč by s touto informací neměly být uloženy další informace, jako je verze operačního systému, který je na stanici nainstalován, „*service pack*“, verze agenta a další. Tyto informace bude zadávat administrátor přes specializované rozhraní, popř. může využít spolupráce s „*active directory*“. Dále musíme vědět, který uživatel v danou dobu na stanici pracoval. Proto musí mít systém informace o pracovnících firmy. Nejdůležitější informací je pochopitelně přihlašovací jméno pracovníka (dále jen login), podle kterého jsme schopni ho identifikovat. Systém tak zjistí, že byl v určitou dobu přihlášen. Samozřejmě je kromě loginu dobré uchovat o takovém pracovníkovi další informace, jako je jméno, adresa, telefon, jeho kancelář, oddělení, kde pracuje, atd. Ve výsledku by pak vše mělo fungovat tak, že si administrátor nechá zobrazit informace o pracovníkovi, např. za poslední týden, a zjistí, které aplikace spouštěl, na kterých to bylo stanicích, jak dlouho v nich byl neaktivní atp.

- **Uložení a přenos dat** – data, která agent na stanici nasbírá, jsou uložena do souboru. V tuto chvíli se nejedná o nic jiného než o, pro běžného uživatele, téměř bezcenná data. Prvním krokem k tomu, aby se z těchto surových dat staly informace, je přenos těchto dat na

kolektor, který je zpracuje a uloží do perzistentní databáze. V této části je rovněž důležité, aby systém pracoval tak, aby v případě přerušení přenosu nedocházelo ke ztrátě dat.

- **Inteligentní prezentace výsledků** – výsledkem celého procesu monitorování musí být prezentace výsledků. Jak již bylo řečeno, součástí systému musí být aplikace, ve které budou inteligentním způsobem zpracovaná data zobrazena. Po zpracování kolektorem se totiž stále jedná pouze o záznamy v databázi, které pořád nejsou snadno čitelné pro člověka. Orientace v těchto datech by sice již nebyla úplně nemožná, ale stále velmi složitá. Prezentace výsledků by proto měla být především jednoduchá a maximálně intuitivní. I ovládání této části systému by mělo být intuitivní do té míry, aby administrátor, který bude se systémem pracovat, vůbec nepotřeboval číst nápovědu apod. Je nutné podotknout, že se tato část vyvíjeného systému značně liší od předchozích zejména proto, že jako jediná přichází do styku s člověkem, zatímco agent a kolektor musí pracovat naprosto autonomně. Prezentace výsledků by měla být prováděna tak, že bude obsahovat formuláře, pomocí kterých si administrátor sestaví dotazy, které mu poskytnou veškeré důležité informace. Systém samozřejmě tyto dotazy bude transformovat na dotazy SQL nad databází.

- **Inteligentní správa systému** – mezi nepostradatelné části systému musí patřit rovněž rozhraní, které bude sloužit pro jeho správu. V tomto případě bude ideální, když bude část systému, sloužící pro správu, součástí aplikace pro prezentaci získaných výsledků. Administrátor musí mít možnost přidávat, editovat a mazat stanice a stejné operace provádět s pracovníky. Je důležité, aby především informace o stanicích byly korektní, protože se podle nich musí řídit kolektor, který se snaží spojit se stanicí při stahování dat. Informace o pracovnících jsou zase důležité kvůli vznikajícím vazbám mezi aktivním oknem a pracovníkem, který toto okno aktivoval (tj. okno se dostalo na popředí). Dále si musíme být vědomi velkého množství informací, které budou získávány, a proto musí být samozřejmou součástí i formulář pro mazání záznamů.

### 2.5.2 Nefunkční požadavky

Nyní si specifikujme nefunkční požadavky systému:

- **Část systému poběží jako služba** – jedním z požadavků je, aby systém běžel jako služba. Prvním důvodem je, že u programu tohoto druhu je bezdůvodné, aby měl uživatelské rozhraní. Druhý důvod je, že nebude žádným způsobem uživatele na stanici rušit nebo

jakkoliv omezovat, takže o něm ani nemusí vědět. Části, které tedy mohou běžet pouze jako služba, budou agent a kolektor, protože část pro prezentaci výsledků se bez uživatelského rozhraní neobejde. Aby však specifikace tohoto požadavku byla úplná, měli bychom si uvědomit, že práce se službou především v čase vývoje je velmi náročná. Proto bude ideální, když bude existovat možnost, že agent i kolektor půjdou spustit jednak jako služby, ale i jako aplikace s uživatelským rozhraním. Při nasazení v praxi by však obě části měly skutečně fungovat pouze jako služby.

- **Platforma .NET a objektová orientace** – důležitým požadavkem je, aby jazyk, ve kterém bude systém implementován byl objektově orientovaný. Důvodem je, že objektově orientovaný systém je při dobrém návrhu a implementaci snáze rozšiřitelný a přepsání implementace do jiného jazyka (též objektově orientovaného) je jednodušší, neboť lze vyjít ze stejného návrhu. Systém by měl běžet na operačním systému Windows XP a platformě .NET. Pokud se bavíme o platformě .NET a objektově orientovaném jazyce, ihned nás napadne, že ideálním jazykem je C#, který byl pro práci na této platformě vytvořen a velmi dobře s ní spolupracuje. S touto otázkou souvisí i zvolený databázový systém MS SQL Server.

- **Inteligentní zpracování dat** – úkolem agenta je monitorovat činnost prováděnou na počítači. Pokud bude systém pracovat ve větší síti, tak agentů bude velké množství a jejich výstupem bude větší množství dat. Proto musí kolektor disponovat schopností inteligentně získaná data zpracovat a uložit do databáze. Inteligentně znamená tak, aby nedocházelo k nekonzistenci mezi získanými daty a administrátor pak při zobrazení prezentace výsledků neměl problémy s pochopením nebo vyhledáním vysledovaných informací.

- **Bezpečnost citlivých dat** – velmi důležitým požadavkem je, aby u dat, která agent získá, byla respektována jejich citlivost. Ačkoliv by se měl pracovník věnovat především své práci, tak jistě bude čas od času docházet k situacím, kdy tomu tak nebude a pak se získané informace mohou stát často velmi osobní. Jsou známy případy z podobných projektů, kde vysledované informace byly natolik citlivé, že na veřejnost skutečně nepatří [10]. Z těchto důvodů by měla být data ukládána jako serializované objekty, které člověk není schopen číst. Součástí projektu by měl být i konceptuální návrh zabezpečení tak, aby ke zneužití údajů nemohlo dojít. Máme na mysli především bezpečný přenos z agenta na kolektor, který bude

implementován pouze nezabezpečeně a ze zabezpečovacích mechanismů bude implementována pouze realizace serializovaných objektů. Případná implementace tohoto návrhu by mohla být otázkou možných rozšíření projektu.

- **Snadná instalace všech součástí systému** – celý systém by měl být snadno a intuitivně použitelný. S tím souvisí samozřejmě kroky, jako je instalace nebo nastavení systému. Jedná se prakticky o čtyři části, které bude nutno instalovat. Agent i kolektor budou mít každý svůj vlastní instalátor. U agenta bychom měli mít možnost konfigurovat dobu, po které bude agent kontrolovat, zda nedošlo na sledované stanici ke změně a dobu, po které se začne považovat pracovní činnost v příslušném okně za neaktivní. U kolektoru bude možnost konfigurovat čas stahování dat z agentů, popř. čas, kdy se má pokusit o stažení znovu, v případě, že první pokus o stažení nebyl úspěšný. U kolektoru musí být nastaven také tzv. „*connectionString*“ sloužící k připojení do databáze, kde bude ukládat data. Jelikož agentů bude mnohem více než kolektorů, tak se bude pravděpodobně konfigurovat agent jen před instalací. Kolektor může být klidně nakonfigurován až po instalaci. Dále by měl být vytvořen intuitivní instalátor pro vytvoření a automatické nakonfigurování databáze. Poslední částí je aplikace pro prezentaci dat, kde však není co instalovat. Pouze je třeba nakonfigurovat zmiňovaný „*connectionString*“, aby systém mohl komunikovat s databází.

# 3 Etická hlediska a právní limity

V červnu roku 1905 uveřejnil snad nejslavnější fyzik všech dob, Albert Einstein, asi nejslavnější rovnici na světě $E = mc^2$ a spolu s ní svoji slavnou teorii relativity. Na počátku jejího vzniku nevyvolala téměř žádný zájem a zůstala tak bez povšimnutí. V dnešní době je jak rovnice, tak teorie relativity základem moderní fyziky. Nejde však pouze o teoretickou fyziku nebo astronomii, kde teorie slouží pro popis vzniku černých děr, počátku či zániku našeho vesmíru nebo pro vysvětlení principu zážehu hvězdy atp. Dnes má využití kupř. i v medicíně, v mnoha lékařských přístrojích, jakým je PET (Positron Emission Tomography) skener sloužící k diagnostice [2].

Uvedli jsme si klady tohoto světového objevu. Nyní si něco povíme o tom, jak lze zdánlivě nezávadnou rovnici využít snad tím nejhorším způsobem. V počátku vzniku rovnice zřejmě ani sám autor Einstein netušil, že by rovnice mohla stát na počátku vývoje takové zbraně, jakou je atomová bomba. Einsteinovy objevy (ale samozřejmě nejen jeho) se staly základem pro ten nejhorší zkázonosný vynález, jaký kdy člověk vynalezl. Slavný fyzik by však jistě nikdy neměl v úmyslu dát do rukou lidem tak destruktivní zbraň. Pramení to i z jeho citátu: [2]

*„Jen dvě věci jsou nekonečné – vesmír a lidská hloupost, tím prvním si ovšem nejsem tak jist.“*

*Albert Einstein*

Ve dnech 6.8. a 9.8. 1945 se ukázalo, že citát je bohužel pravdivý. V osudových dnech došlo ke svržení dvou atomových pum na japonská města Hirošima a Nagasaki. Následky obou výbuchů byly naprosto ničivé a kromě mnoha obětí zůstaly následky v těchto městech až do dnešních dob.

Důvodem, proč jsme natolik odbočili od tématu je, abychom poukázali na fakt, že nelze o ničem prohlásit, zda je to jednoznačně užitečné, nebo zda to „něco“ může dokonce přinášet i zkázu a neštěstí. Záleží jen a pouze na tom, v jakých rukou se to nachází a jaké máme s danou věcí úmysly. Řetězová reakce se využívá v jaderných elektrárnách a přináší nám energii, která je převáděna na energii elektrickou. A jak již bylo řečeno existuje neřízená řetězová reakce, která má tu schopnost, že může navždy změnit naše životy.

Ve spojení s tématem bezpečnosti nebo nebezpečnosti použití určité věci bychom se mohli zmínit o Philu Zimmermannovi a jeho příběhu z doby 80. a 90. let. Ačkoliv se nám to může zdát naprosto neetické a nemorální, v té době byly naprosto běžné odposlechy telefonních linek a informace po Internetu putovaly nezabezpečeně, nebo s takovou bezpečností, kdy byly jisté vládní

instituce schopny tyto informace bez větších problémů dekódovat. Trnem v oku těmto institucím se stal algoritmus RSA (pojmenovaný po prvních písmenech příjmení svých autorů – **R**ivest, **S**hamir, **A**dleman) a již zmíněný Phil Zimmerman. RSA je šifra asymetrická (což znamená, že klíč pro kódování a dekódování není stejný) a především je to šifra v reálném čase nerozluštitelná. Nerozluštitelná je, pokud jsou dodrženy určité předpoklady, které souvisí se složitostí dešifrování. RSA je totiž teoreticky rozluštitelná, ovšem v současné době k tomu neexistuje žádná technika, která by toho byla schopna v reálném čase [3].

Vláda chtěla mít možnost dekódovat téměř každou komunikaci, kterou lidé využívali a hájila se národní bezpečností. Zejména šlo o boj proti terorismu a organizovanému zločinu. Phil Zimmermann byl ovšem toho názoru, že nárok na soukromí by měl mít každý bez výjimky. Vytvořil proto program s názvem PGP (Pretty Good Privacy). Šlo o jakýsi balík šifrovacího softwaru, který byl schopen provést velmi rychlé a efektivní šifrování pomocí algoritmu RSA. Klasické asymetrické šifrování trvalo na osobních počítačích velmi dlouho, a proto použil Zimmermann trik, kde kombinoval asymetrické a symetrické šifrování. Použil symetrickou šifru v té době známou jako IDEA (International Data Encryption Algorithm), která je podobná dnešnímu DES (Data Encryption Standard) algoritmu. Touto šifrou program zašifroval samotnou zprávu. U symetrické šifry je však ještě třeba přenést klíč. Zde přichází na řadu algoritmus RSA, kterým je tento klíč zašifrován a tím pádem bezpečně poslán. Program PGP měl i další funkce jako jsou generování klíčů, elektronický podpis atp. [3].

Svůj produkt Zimmermann dokončil a stál před problémy, z nichž ani jeden nebyl technického směru. RSA byl totiž patentovaný algoritmus společností RSA Data Security, Inc. a pro uvedení na trh musel mít od této společnosti licenci. Jelikož ale PGP mělo sloužit jednotlivcům a nikoliv podnikům, tak se Zimmermann rozhodl tento problém ignorovat. Větší problém pro něj představovala vláda a senátní návrh trestního zákona z roku 1991, ve kterém se mluvilo o tom, že veškerá elektronická komunikace musí zaručovat, že vládní úřady budou mít možnost získat informace z těchto komunikací v otevřené podobě. To se samozřejmě rozcházelo s filozofií programu PGP i s názorem Phila Zimmermanna. Celá věc nakonec vyústila tak daleko, že vláda Zimmermanna obvinila z nelegálního exportu zbraní, protože šifrovací software zahrnula do definice zbraní a Zimmermann jej dal na Internet, kde si ho mohl stáhnout naprosto každý. Vše se táhlo ještě mnoho let a debata v podstatě není uzavřena dodnes. Hlavní otázkou zůstává, zda by měla být povolena neomezená kryptografie [3].

Předchozí odstavce o Albertu Einsteinovi a atomové bombě a o Philu Zimmermannovi a jeho PGP jasně ukazují, že nic na světě není pouze černé nebo bílé. Fakt, že vlády různých zemí jsou

schopny odposlouchávat komunikace mezi lidmi, zcela jistě napomáhá k potlačení zločinnosti a k infiltrování organizovaných skupin, mezi které mohou patřit např. i teroristické skupiny. Se stejnou jistotou ovšem můžeme tvrdit, že podobné odposlechy komunikace jsou zneužívány pro účely, které naprosto nesouvisejí s bezpečností státu a jejich občanů. Katastrofální následky by mohl mít fakt, že pokud má vláda schopnost odposlouchávat libovolnou komunikaci, tak existuje možnost, že tuto schopnost bude mít i osoba či organizace, která stojí na opačné straně „bojiště". A pokud tato možnost existuje, měli bychom ji brát v úvahu a vynaložit maximální úsilí pro to, abychom nic podobného nedopustili. Metaforicky řečeno i informace ve špatných rukou se mohou stát atomovou bombou. V následující podkapitole se více zaměříme na etiku a morálku v kontextu naší aplikace.

## 3.1 Etika systému monitorování uživatelů

Náš software pro sledování aktivit uživatelů by měl fungovat tak, že mu neunikne sebemenší činnost pracovníka na příslušném počítači. Cílem tohoto sledování je vytvořit obraz o tom, jak který pracovník v podniku pracuje. Měl by odhalovat „černé ovce", tudíž pracovníky, kteří zdaleka neplní své povinnosti tak, jak by měli. V dnešním světě si už ani neuvědomujeme, jak často nás sledují kamery v obchodech, ve školách, v práci apod. Zvykli jsme si a považujeme tuto skutečnost za samozřejmost. Ani nás nenapadne namítat, že je tím omezováno naše soukromí nebo cokoliv podobného. Pokud bychom však někomu pověděli, že má pracovat na počítači a veškerá činnost, kterou provede bude pečlivě sledována, zřejmě by to nepřijal s velkým nadšením. Zejména bychom velké rozhořčení čekali od pracovníků, kteří pracují ve firmě již delší dobu a nejsou na tento přístup zvyklí. Nově přijatý pracovník by to spíš považoval za něco, co je v podniku naprosto běžné a co musí přijmout jako fakt, pokud chce ve firmě pracovat.

Stinnou stránkou zůstává fakt, že program lze využít i ne zcela etickým způsobem a za jistých okolností by se dal kategorizovat jako „*malware*" (obecné označení pro škodlivý software). Kupříkladu trojské koně, jejichž název pochází z řecké mytologie, mají mnoho společných prvků chování jako náš program. Především je pro neznalé uživatele složitější odhalit jeho přítomnost v počítači. Sleduje činnost uživatele, aniž by o tom uživatel věděl. Může tím např. zjistit, které webové stránky navštěvuje nejčastěji a vytvářet tak půdu pro jiný „*malware*" zvaný „*spam*". V neposlední řadě se musí připojovat k jinému počítači, samozřejmě opět bez vědomí uživatele, aby mohl zaslat nasbíraná data o činnosti na dané stanici.

Měli bychom se zamyslet nad tím, zda existuje způsob, jak zabránit nebo alespoň omezit zneužití takového systému. Pokud však mluvíme o zneužití systému, musíme si uvědomit, že jde o systém určený k zjišťování informací. Proto nejde v pravém slova smyslu o zneužití systému, protože samotným zneužitím by mohlo být jeho správné používání. Co však lze zneužít, to jsou získané informace. Bohužel zřejmě neexistuje způsob, jak se proti tomuto zneužití informací efektivně bránit. Vezmeme-li v úvahu lidský faktor, je to nemožné zcela určitě.

Kevin Mitnick, který je považován za nejslavnějšího hackera světa ve své knize „Umění klamu“[5] vysvětluje a na mnoha příkladech dokazuje, že nejslabším článkem jakéhokoliv systému (ne nutně počítačové sítě apod.) je vždy člověk. Můžeme se sebevíc snažit o zabezpečení souboru na straně stanic, snažit se provést bezpečný přenos na kolektor a uložení do databáze, ale koncovým bodem je administrátor. Pokud ho kdokoliv přesvědčí, že má některá data zaslat např. na neznámý email, je to přesně příklad jako vystřižený z Mitnickovy knihy.

## 3.2 Soulad se zákonem

Bavíme-li se o spojitosti etiky s tímto projektem, musíme vzít v potaz i zákony České republiky. Již jsme se okrajově zmínili o zákoně č.140/1961 Sb. §239 a §240 v platném znění, částka 65/1961 o porušování tajemství dopravovaných zpráv [13], který má následující znění:

§ 239 (1) Kdo úmyslně poruší tajemství

**a)** uzavřeného listu nebo jiné písemnosti, při poskytování poštovní služby nebo jiným dopravním zařízením, nebo

**b)** zprávy podávané telefonem, telegrafem nebo jiným takovým veřejným zařízením,

bude potrestán odnětím svobody až na šest měsíců.

§ 239 (2) Pracovník provozovatele poštovních služeb nebo telekomunikační služby, který

**a)** spáchá čin uvedený v odstavci 1,

**b)** jinému úmyslně umožní spáchat takový čin, nebo

**c)** pozmění nebo potlačí písemnost obsaženou v poštovní zásilce nebo dopravovanou dopravním zařízením anebo zprávu podanou telefonicky, telegraficky nebo dopravovanou podobným způsobem,

bude potrestán odnětím svobody až na jeden rok nebo zákazem činnosti.

§ 240 (1) Kdo v úmyslu způsobit jinému škodu nebo opatřit sobě nebo jinému neoprávněný prospěch

**a)** prozradí tajemství, o němž se dozvěděl z písemnosti, telegramu nebo telefonního hovoru, které nebyly určeny jemu, nebo

**b)** takového tajemství využije,

bude potrestán odnětím svobody až na jeden rok.

§ 240 (2) Pracovník poštovní nebo telekomunikační služby, který

**a)** spáchá čin uvedený v odstavci 1, nebo

**b)** jinému úmyslně umožní spáchat takový čin,

bude potrestán odnětím svobody až na dvě léta nebo zákazem činnosti.

Taková je tedy ukázka zákona, které často známe pod názvem listovní tajemství. Náš systém odhalí, že pracovník psal email, ovšem není uzpůsoben k tomu, aby zjistil komu tento email psal a v žádném případě není schopen sledovat obsah emailu. Jediné, co zjistí, je předmět tohoto emailu a to pouze a jenom v případě, že pracovník použije poštovního klienta, který zobrazuje předmět zprávy v titulku okna (př. Mozilla Thunderbird). Proto můžeme prohlásit, že tento projekt se nedostává do konfliktu se zákonem o porušování tajemství dopravovaných zpráv.

Druhým zákonem, který je již z našeho pohledu mnohem zajímavější nese název „*Ochrana majetkových zájmů zaměstnavatele a ochrana osobních práv zaměstnance*“, který je součástí zákona č. 262/2006 Sb. § 316 v platném znění, částka 85/2006, jehož znění je následující [14]:

§ 316 (1) Zaměstnanci nesmějí bez souhlasu zaměstnavatele užívat pro svou osobní potřebu výrobní a pracovní prostředky zaměstnavatele včetně výpočetní techniky ani jeho telekomunikační zařízení. Dodržování zákazu podle věty první je zaměstnavatel oprávněn přiměřeným způsobem kontrolovat.

§ 316 (2) Zaměstnavatel nesmí bez závažného důvodu spočívajícího ve zvláštní povaze činnosti zaměstnavatele narušovat soukromí zaměstnance na pracovištích a ve společných prostorách zaměstnavatele tím, že podrobuje zaměstnance otevřenému nebo skrytému sledování, odposlechu a záznamu jeho telefonických hovorů, kontrole elektronické pošty nebo kontrole listovních zásilek adresovaných zaměstnanci.

§ 316 (3) Jestliže je u zaměstnavatele dán závažný důvod spočívající ve zvláštní povaze činnosti zaměstnavatele, který odůvodňuje zavedení kontrolních mechanismů podle odstavce 2, je zaměstnavatel povinen přímo informovat zaměstnance o rozsahu kontroly a o způsobech jejího provádění.

Tento paragraf má ještě čtvrtý odstavec, který však s naší problematikou nesouvisí. Ostatní tři odstavce se však tváří zcela jinak. Zejména v prvním odstavci se píše, že „*dodržování zákazu podle věty první je zaměstnavatel oprávněn přiměřeným způsobem kontrolovat*“, což je dostatečný argument pro nasazení podobného systému ve firmě. Druhý odstavec hovoří o tom, že zaměstnanec nesmí být sledován, což by jistě značně narušovalo jeho soukromí. Vyvíjený systém je však vystavěn tak, že nesleduje pracovníka, jako bychom si to mohli představit např. u kamerového systému, ale pouze sleduje jeho činnost, čímž zcela jistě vhodným způsobem hlídá dodržování zákazu ve smyslu prvního odstavce. Cílem nasazení tohoto nebo podobného systému ve firmě není vytváření zbytečných konfliktů mezi zaměstnanci a zaměstnavateli, ale spíše naplnění rčení „*důvěřuj, ale prověřuj*“. V každém případě by měli být pracovníci vždy vhodnou formou (např. vydáním příslušných směrnic) informováni o existenci podobného systému v jejich firmě a v práci se to tak skutečně řeší.

## 3.3 Závěr a diskuze

Shrneme-li tuto kapitolu, tak musíme závěrem říci, že náš program, resp. informace, které získá mohou být velmi užitečné, pokud jsou využívány k tomu, k čemu jsou skutečně určeny. Z pohledu návrháře systému si však nemůžeme být jisti, že tomu tak skutečně bude. Můžeme si pouze přát, aby systém přinášel užitek a z pohledu etiky byl využíván co nejetičtěji s ohledem na lidské soukromí a ochranu osobnosti. S tím pochopitelně souvisí i dodržování výše probíraných paragrafů.

# 4 Modely a architektura systému

Před samotnou implementací systému je nutné vytvořit potřebné modely, které celou implementaci velmi usnadní. Dalo by se říct, že vývoj softwaru lze rozdělit na proces analýzy a proces syntézy. Analýza je ta část, které se doposud věnujeme. Snažíme se určit, co přesně by měl systém vykonávat a postupně přecházíme k tomu, jak by to měl systém vykonávat s ohledem na požadavky, které na něj klademe. Když jsou vytvořeny modely, přichází ke slovu implementace jednotlivých částí systému.

Během implementace se začínají jednotlivé části systému spojovat, což je proces syntézy. Výsledkem by měl být systém, který co nejvíce koresponduje s vytvořenými modely a maximálně splňuje veškeré vytvořené požadavky. Tento proces se využívá z toho důvodu, že díky modelování si můžeme včas povšimnout celé řady nedostatků nebo chyb v návrhu, které jsou zatím v raném stádiu a jejich odstranění je pouze otázkou změny návrhu. Pokud bychom byli nuceni podobný problém odstraňovat již v implementovaném systému, mohlo by se jednat o náročnou záležitost, zejména z důvodu spojitostí a návazností těžko separovatelných částí systému.

## 4.1 Modelování případů použití

Mezi modely, které se snaží specifikovat požadavky na systém, patří model případů použití. Často se užívá zejména pro jeho silný popisný charakter a jeho snadnou pochopitelnost. Diagram případů použití určuje, co by měl systém dělat, ale neřeší jak by to měl udělat. Velmi často se tento diagram využívá při komunikaci se zákazníkem, nebo obecněji řečeno, s osobou, která se příliš neorientuje v oblasti informačních technologiích.

Na obr. 1 vidíme diagram případů použití našeho vyvíjeného systému. Ačkoliv je většina částí diagramu poměrně snadno pochopitelná, pokusíme se nyní zaměřit na ty méně intuitivní části. Hned ve druhé kapitole jsme si definovali dvě role, které nás budou v našem systému zajímat. Byli to administrátor a pracovník. V diagramu však vidíme, že se v něm pracovník vůbec nevyskytuje. Je to proto, že ve skutečnosti žádná interakce mezi pracovníkem a systémem neexistuje. To plyne i z toho, že systém může běžet na stanici, kde je pracovník, aniž by o tom měl tušení. Důvodem, proč jsme definovali pracovníka jako jednu z rolí, je, že ve chvíli, kdy „abstrahujeme“ o úroveň výš z našeho systému, zjistíme, že pracovník je součástí tohoto systému jakožto celku, ale není nijak zainteresován z pohledu vývoje systému. Je však nezbytně nutné tuto roli definovat (viz str. 5), pro vysvětlení

funkce systému z vnějšího pohledu. Naproti tomu je role administrátora v porovnání s pracovníkem nedílnou součástí systému a jako takový musí být i modelován.


***Obr. 1*** *– Diagram případů použití*

Je snadné si všimnout, že tento konkretní model tvoří strom (vždy tomu tak není, může tvořit obecný graf s kružnicemi), jehož kořenem je administrátor. Kromě toho vidíme, že mezi uzly (vyjma kořenu), zde existuje relace generalizace, resp. specializace. Příkladem je nastavení parametrů, které

se dále větví na nastavení agenta a kolektoru, které se opět větví na další případy. Pokud bychom nechtěli tuto relaci využít a rovnou bychom ke kořenu vázali konkrétní případy použití, diagram by byl značně nepřehledný a postrádal by jakousi logickou strukturu. Popišme si nyní význam některých konkrétních případů použití:

- **Nastavení doby intervalu sledování** – agent musí fungovat tak, že bude v určitém časovém intervalu zjišťovat, co se na stanici změnilo od poslední kontroly, tzn. která aplikace je nyní na popředí, které procesy byly zrušeny apod. Proto tento časový interval bude možno nastavit.

- **Nastavení doby sledování začátku neaktivity** – jde o nastavení času, který musí uplynout od doby, kdy byl uživatel naposledy aktivní, aby agent jeho činnost začal zaznamenávat jako neaktivitu.

- **Nastavení časového intervalu pro přenos dat** – jde o nastavení doby, která musí uplynout mezi jednotlivými úspěšnými pokusy o stažení dat z agenta na kolektor.

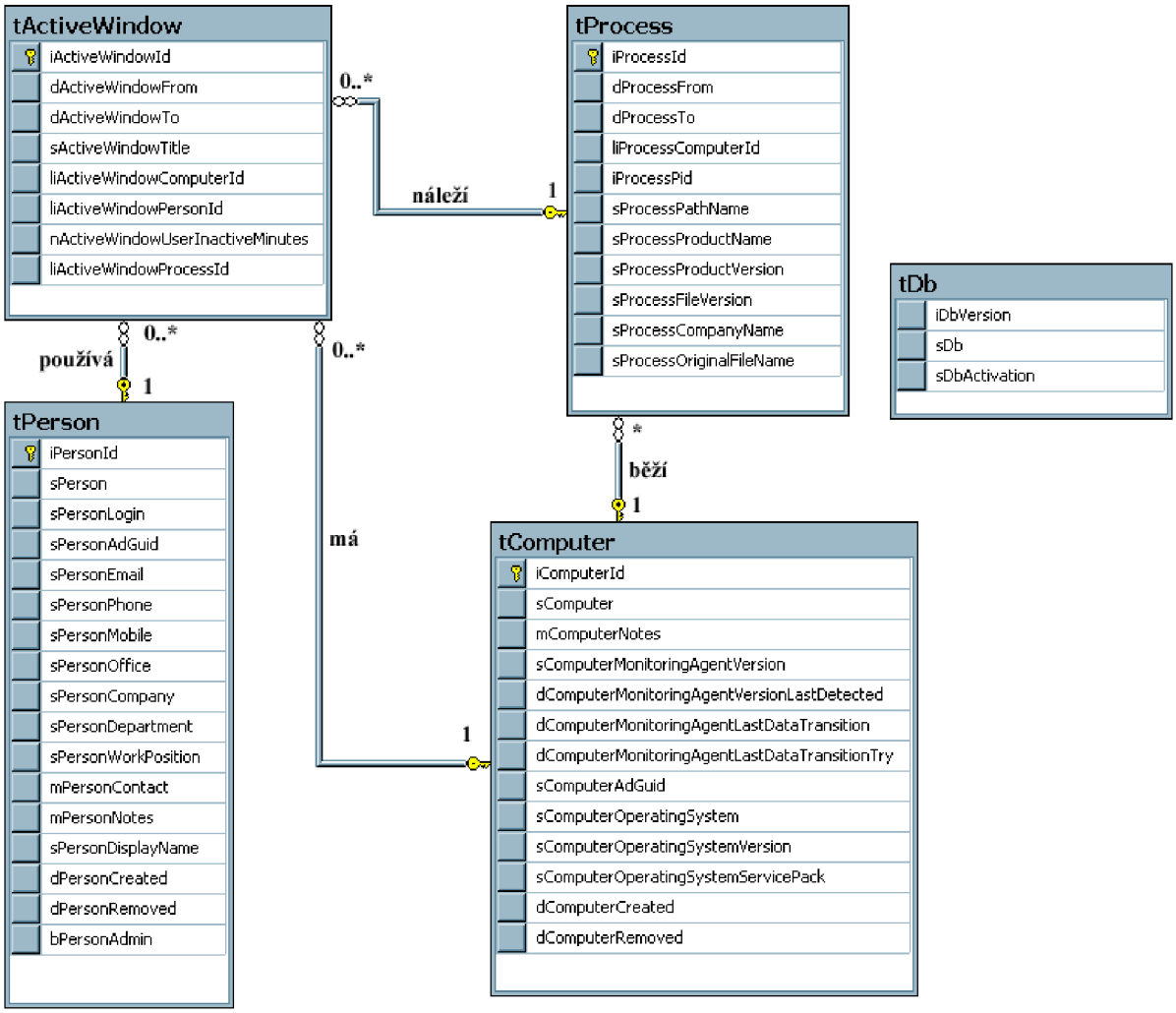
- **Nastavení časového intervalu pro přenos dat po neúspěšném přenosu** - jde o nastavení doby, která musí uplynout po neúspěšném pokusu o stažení dat z agenta na kolektor, abychom se znovu pokusili je získat.

Výjimečné uzly modelu jsou zejména případy použití: „zjišťování využití SW“ a „prohlížení získaných dat“. V prvním případě je vazba k dalším uzlu (listu) označena „<< *include* >>“, což vyjadřuje, že pokud chce administrátor zjistit, jak je využíván software nainstalovaný na stanicích, musí vyplnit formulář, kterým systému řekne, co přesně chce zjišťovat. Naopak „<< *extend* >>“ označuje vazby u prohlížení získaných dat a vyjadřuje, že administrátor může prohlížet získaná data, ale kromě toho může využít vyhledávacího formuláře nebo položky různě řadit.

## 4.2 Modelování databáze

Modelováním databáze máme na mysli znázornění uložení a provázání perzistentních dat v relační databázi. K tomuto účelu vytvoříme entitně relační diagram (dále jen ER diagram) a podrobně si popíšeme význam jeho entit, atributů a vazeb mezi entitami. Jen pro úplnost si doplníme názvosloví.

Entitami nazýváme objekty reálného světa, které jsou zpravidla transformovány do tabulek relačních databází. Tyto entity obsahují atributy, které korespondují se sloupci vytvářených tabulek. Vazby mezi entitami popisují, v jakém vztahu jsou mezi sebou tyto entity. Na obr. 2 můžeme vidět vztahy mezi tabulkami vyjádřené pomocí ER diagramu.




***Obr. 2*** *– ER diagram*

Pokud mluvíme o modelu databáze, měli bychom se alespoň okrajově zmínit o tzv. normálních formách a normalizacích. Jedná se o to, že díky normalizaci databáze se vyvarujeme přílišné složitosti, zbytečnému opakování dat (tzv. redundanci dat) a chybám, které mohou mezi daty vznikat a nemusí být na první pohled zřejmé. U normalizované databáze pak také můžeme vytvářet efektivnější a jednodušší dotazy. Nebudeme se zabývat konkrétními typy normalizace, pouze si řekneme, že v praxi je běžné, že jsou tabulky normalizovány na třetí normální formu (dále jen 3.NF). Naše databáze je ale pouze ve druhé normální formě (dále jen 2.NF). Přesněji řečeno tabulky „*tPerson*“ a „*tComputer*“ jsou ty, které se nacházejí pouze ve 2.NF. Důvodem je, že počet řádků těchto tabulek nebude nijak velký, a proto se nevyplatí je dále normalizovat. V případě, kdy máme např. internetový obchod, kde se může vyskytovat mnoho miliónů záznamů, je téměř nutné dosáhnout 3.NF, ovšem pro náš projekt je 2.NF naprosto dostačující.

Jednotlivé entity a jejich atributy si popíšeme podrobněji v následujících tabulkách:

| **Tabulka tActiveWindow** | | | |
|---|---|---|---|
| **Název atributu** | **Popis** | **Typ** | **Příklad** |
| iActiveWindowId | Primární klíč tabulky | Integer (PK) | 114 |
| dActiveWindowFrom | Čas, od kdy bylo okno aktivní | Datetime | 27.9.2007 16:03:25 |
| dActiveWindowTo | Čas, do kdy bylo okno aktivní | Datetime | 27.9.2007 16:03:43 |
| sActiveWindowTitle | Nadpis okna | Varchar(255) | Seznam – Mozzila Firefox |
| liActiveWindowComputerId | Cizí klíč do tabulky tComputer | Integer (FK) | 1 |
| liActiveWindowPersonId | Cizí klíč do tabulky tPerson | Integer (FK) | 1 |
| nActiveWindowUserInactive Minutes | Doba, po kterou byl uživatel v okně neaktivní | Float | 0,133 |
| liActiveWindowProcessId | Cizí klíč do tabulky tProcess | Integer (FK) | 89 |

| **Tabulka tProcess** | | | |
|---|---|---|---|
| **Název atributu** | **Popis** | **Typ** | **Příklad** |
| iProcessId | Primární klíč tabulky | Integer (PK) | 77 |
| dProcessFrom | Začátek procesu | Datetime | 27.9.2007 9:54:15 |
| dProcessTo | Konec procesu | Datetime | 27.9.2007 10:28:21 |
| liProcessComputerId | Cizí klíč do tabulky tComputer | Integer (FK) | 1 |
| iProcessPid | ID procesu (Process ID) | Integer | 3920 |
| sProcessPathName | Adresářová cesta k souboru, který | Varchar(1024) | C:\xampp\apache\bin\ap |

| | | | |
|---|---|---|---|
| | proces spustil | | ache.exe |
| sProcessProductName | Jméno produktu | Varchar(1024) | Apache HTTP Server |
| sProcessProductVersion | Verze produktu | Varchar(255) | 2.2.8 |
| sProcessFileVersion | Verze souboru | Varchar(255) | 2.2.8 |
| sProcessCompanyName | Jméno výrobce produktu | Varchar(1024) | Apache Soft. Foundation |
| sProcessOriginalFileName | Originální jméno souboru | Varchar(1024) | httpd.exe |

| **Tabulka tDb** | | | |
|---|---|---|---|
| **Název atributu** | **Popis** | **Typ** | **Příklad** |
| iDbVersion | Verze databáze | Integer | 3 |
| sDb | Popis databáze | Varchar(255) | alc.monitoring |
| sDbActivation | Aktivační klíč | Varchar(255) | ALC,spol. s r.o./J1Y2NEB53HBVI8G4K |

| **Tabulka tPerson** | | | |
|---|---|---|---|
| **Název atributu** | **Popis** | **Typ** | **Příklad** |
| sPersonId | Primární klíč tabulky | Integer (PK) | 2 |
| sPerson | Jméno pracovníka | Varchar(255) | Aleš Skopal |
| sPersonLogin | Login pracovníka | Varchar(255) | xskopa13 |
| sPersonAdGuid | Identifikátor pracovníka | Varchar(255) | xskopa13 |
| sPersonEmail | Email | Varchar(255) | xskopa13@stud.fit.vutbr.cz |
| sPersonPhone | Telefon | Varchar(255) | +420 534 229 020 |
| sPersonMobile | Mobilní telefon | Varchar(255) | +420 776 792 063 |
| sPersonOffice | Kancelář | Varchar(255) | P029 |
| sPersonCompany | Společnost | Varchar(255) | VUT FIT |
| sPersonDepartment | Oddělení | Varchar(255) | Fakulta Infomačních systémů |
| sPersonWorkPosition | Pracovní pozice | Varchar(255) | Student |
| mPersonContact | Kontakt | Text | Aleš Skopal, Bezručova 1115, 664 34 Kuřim |
| mPersonNotes | Poznámka | Text | Velmi pracovitý |
| sPersonDisplayName | Zobrazované jméno | Varchar(255) | xskopa13 |
| dPersonCreated | Čas vytvoření pracovníka v systému | Datetime | 27.9.2007 14:46:59 |
| dPersonRemoved | Čas zrušení pracovníka v systému | Datetime | 26.11.2008 7:28:13 |
| bPersonAdmin | Příznak pro administrátora | Bit | 0 |

| Tabulka tComputer | | | |
|---|---|---|---|
| **Název atributu** | **Popis** | **Typ** | **Příklad** |
| iComputerId | Primární klíč tabulky | Integer | 1 |
| sComputer | Název počítače | Varchar(255) | FIT-MACHINE |
| mComputerNotes | Popis počítače | Text | Školní počítač |
| sComputerMonitoringAgentVersion | Verze agenta na počítači | Varchar(255) | 8.0.50727.42 |
| dComputerMonitoringAgentVersionLastDetected | Čas poslední detekce agenta | Datetime | 27.9.2007 16:01:55 |
| dComputerMonitoringAgentLastDataTransition | Čas posledního přenosu dat z počítače | Datetime | 27.9.2007 16:08:39 |
| dComputerMonitoringAgentLastDataTransitionTry | Čas posledního pokusu o přenos dat z počítače | Datetime | 27.9.2007 16:08:38 |
| sComputerAdGuid | Identifikátor počítače | Varchar(255) | FITPC001 |
| sComputerOperatingSystem | Operační systém | Varchar(255) | Windows XP Professional |
| sComputerOperatingSystemVersion | Verze OS | Varchar(255) | Version 2002 |
| sComputerOperatingSystemServicePack | Service pack | Varchar(255) | Service Pack 2 |
| dComputerCreated | Čas vytvoření počítače | Datetime | 27.9.2007 14:50:23 |
| dComputerRemoved | Čas zrušení počítače | Datetime | 11.5.2008 11:34:57 |

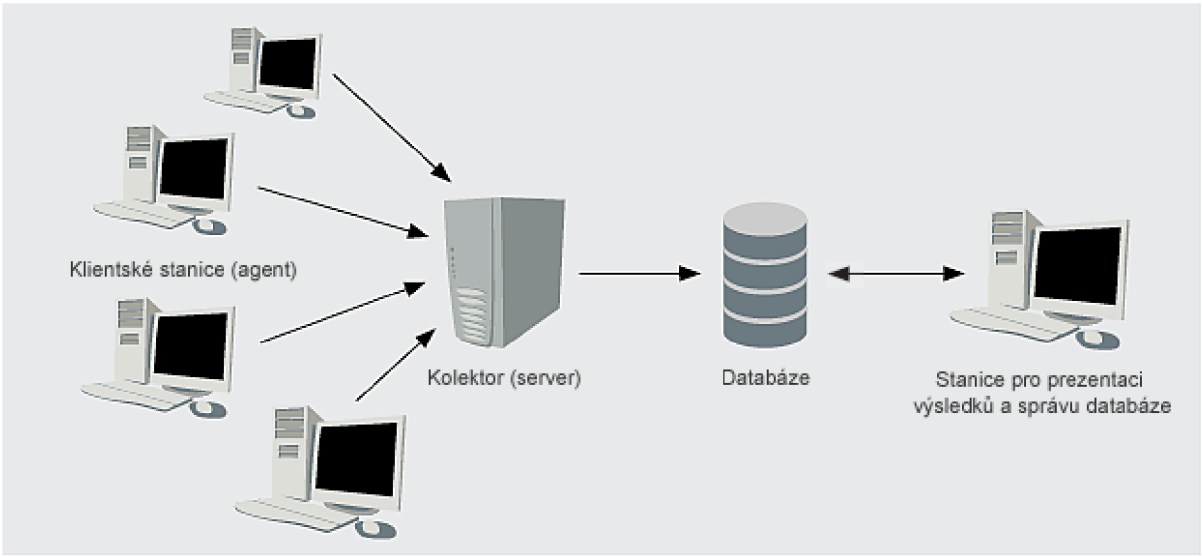
Tímto jsme popsali konkrétní atributy všech entit vyskytujících se v našem ER diagramu. Zbývá nám vysvětlit si vztahy mezi těmito entitami. Systém Windows XP je založený na práci s okny, a proto musí existovat vazba mezi entitami „*tComputer*“ a „*tActiveWindow*“. V diagramu jsme ji popsali slovesem „*má*“. Tato vazba vyjadřuje, že okno je aktivní vždy na nějakém konkrétním počítači. Tato okna jsou využívána pracovníky, a proto máme vazbu „*používá*“ mezi entitami „*tActiveWindow*“ a „*tPerson*“. Okna však vzniknou díky procesu, ke kterému jsou vázána vazbou „*náleží*“ mezi entitami „*tActiveWindow*“ a „*tProcess*“. Musíme si ale uvědomit, že ne všechny procesy mají své okno, proto je u entity „*tActiveWindow*“ kardinalita „*0..**“. Poslední vazba s názvem „*běží*“ je mezi entitami „*tProcess*“ a „*tComputer*“, která vyjadřuje, že procesy vždy běží na nějakém konkrétním počítači.

## 4.3 Architektura systému

Architekturu systému můžeme popsat mnoha způsoby a z mnoha pohledů. Doposud jsme popisovali systém poměrně abstraktními prostředky, jako jsou modely nebo různé metody popisu. Ačkoliv u abstraktních popisů zůstaneme, fyzickou architekturu systému si v následující kapitole popíšeme pomocí fyzických prostředků, se kterými se běžně setkáváme. O něco méně zřejmý bude popis vrstev systému, který má však skvělý popisný charakter z pohledu vytváření, přenosu a zpracování dat.

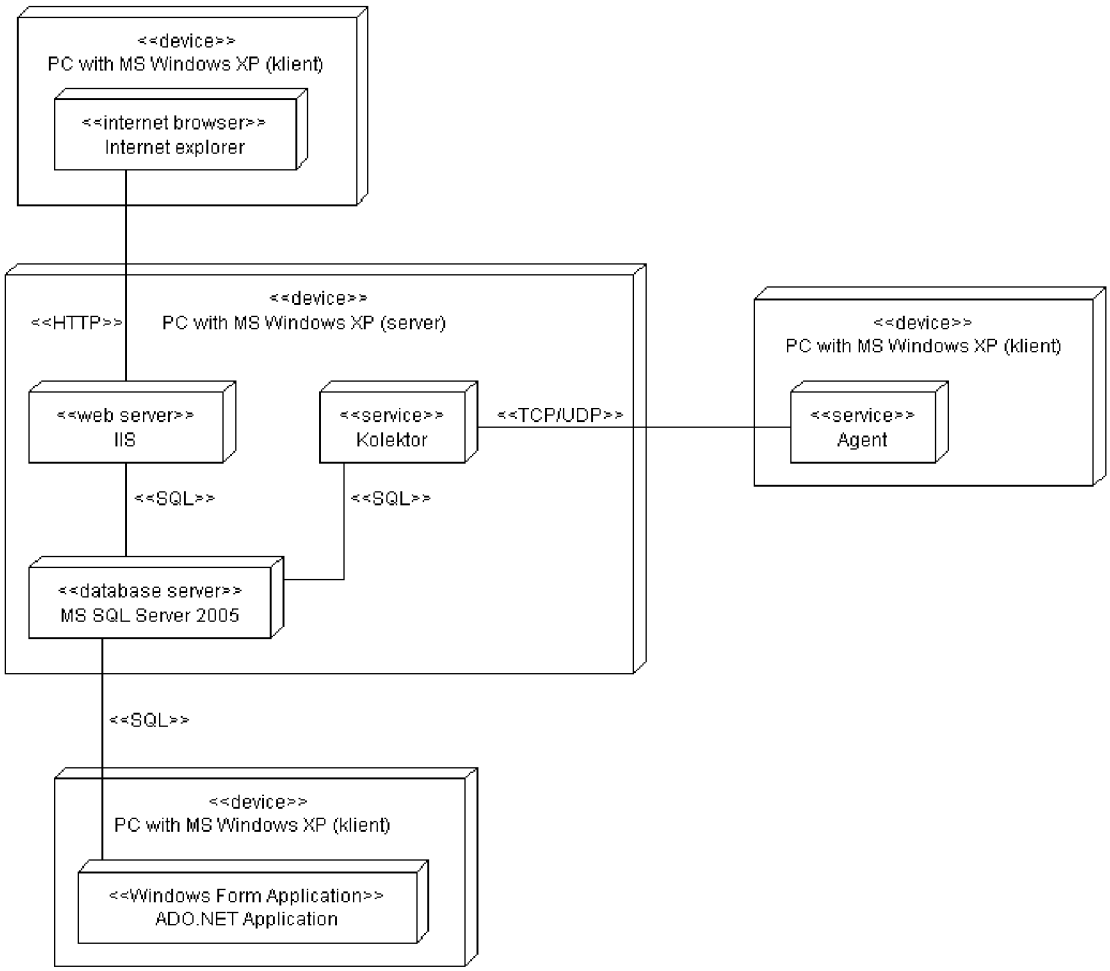
### 4.3.1 Fyzická architektura systému

Chceme-li modelovat fyzickou architekturu systému, zahrnujeme do ní především hardware. Znázorněný systém můžeme vidět na obr. 3. Základem je počítačová síť tvořená pracovními stanicemi. Samozřejmě, že v počítačové síti existuje mnoho dalších prvků, jako jsou směrovače, síťové tiskárny apod., ale nás zajímají jenom a pouze pracovní stanice. Zanedbejme proto zapojení v síti a prvky, které by pro síť byly nezbytné. Na každé z těchto stanic je nainstalován agent, který monitoruje činnost na dané stanici. Jakmile kolektor pošle žádost o získaná data, jsou tato data přenesena na kolektor, který je zpracovává a ukládá do databáze. Tato data jsou dále zpracována aplikací běžící na stanici určenou pro prezentaci výsledků. Ačkoliv není na obrázku kolektor znázorněn jako stanice, je nutno dodat, že kolektor (jakožto služba) patrně bude součástí stanice pro prezentaci výsledků a správu databáze.




***Obr. 3** – Fyzická architektura systému*

Obr. 3 nám zobrazuje architekturu z pohledu hardwaru. Je z něj patrné, jak bude přibližně vypadat síť, ve které bude systém nasazen. Nyní bychom měli vytvořit diagram, který ukazuje nejen fyzický hardware, ale i software, který je na tomto hardwaru nasazen [1]. Takový diagram nazýváme diagramem nasazení a můžeme ho vidět na obr. 4.




***Obr. 4*** *– Diagram nasazení*

V diagramu vidíme tři klienty a jeden server. Prvním klientem je náš agent, který bude monitorovat stanici. Druhým klientem bude ADO.NET aplikace nebo ASP.NET stránky, tedy část systému sloužící pro zobrazení výsledků a správu databáze. Z hlediska funkce systému nehraje žádnou roli, zda pro tuto prezentaci a správu vytvoříme intranetový portál nebo aplikaci s uživatelským rozhraním, a proto jsou v diagramu zahrnuty obě možnosti. Situaci lze přirovnat k

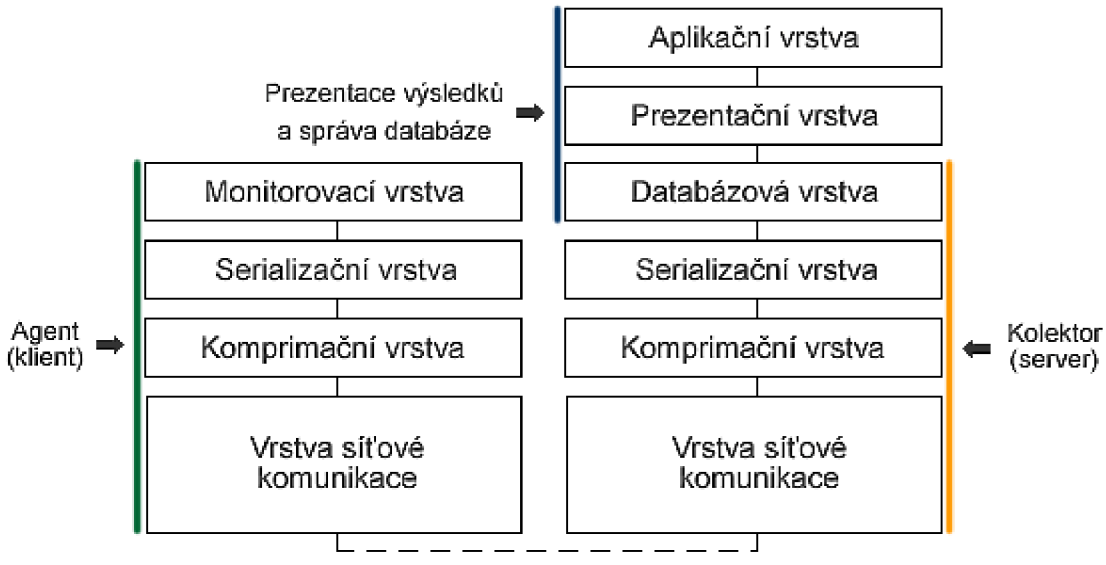
internetovým stránkám, na které přistupuje v jeden okamžik více uživatelů a to zcela nezávisle na sobě. Z toho plyne, že i v tomto případě můžeme přistupovat k výsledkům z více míst současně.

Velmi důležitou částí je server, na kterém běží kolektor. Součástí serveru je databáze SQL a rozhraní, přes které se do databáze přistupuje. Rozhraní má název „*SQL Server .NET Data Provider*“. V případě, že budeme pro přístup k datům používat webové rozhraní, musí být součástí serveru i webový server IIS. Vazby mezi jednotlivými komponentami jsou vcelku jednoznačné. Agent nasbíraná data posílá přes TCP nebo UDP spojení na kolektor, který je zpracuje a pomocí dotazů SQL uloží do databáze. Potom může administrátor pomocí ADO.NET aplikace, resp. pomocí webového rozhraní, přistupovat přes SQL, resp. přes http k získaným datům. Úkolem aplikace, resp. webového rozhraní, je zajištění inteligentní prezentace dat a poskytnutí rozhraní pro správu databáze.

Mohlo by být matoucí, že i když oba diagramy znázorňují architekturu systému, je vidět, že příliš silná vazba mezi nimi není. Na obou vidíme prvky jako agent, kolektor, databáze a klient (stanice) pro prezentaci výsledků a správu databáze, ale jejich rozložení je pokaždé jiné. Je třeba pochopit, že ačkoliv oba obrázky popisují stejný systém a obsahují stejné prvky, každý popisuje systém z jiného pohledu a v jiné souvislosti.

### 4.3.2 Architektura vrstev systému

Nyní se zaměříme na architekturu z pohledu vrstev systému, kdy se pro změnu trochu vzdálíme od reálných fyzických prostředků, které jsme používali v předchozí kapitole.




***Obr. 5** – Architektura vrstev*

Ukážeme si rozdělení systému na vrstvy. Obr. 5 znázorňuje architekturu vrstev našeho systému. Pokud bychom chtěli nějak vyjádřit vazbu s diagramem na obr. 3 (viz str. 26), usnadní nám to barevné znázornění na obr. 5. Čtyři vrstvy v levé části obrázku označené zeleným pruhem popisují monitorovacího agenta, který je nasazen na stanici. První část, označená oranžovým pruhem, je kolektor, který sbírá data ze stanic. Databázová vrstva by mohla korespondovat s databází, a jak je vidět, i v tomto diagramu (podobně jako na obr. 3) je přístupná ze dvou stran – ze spodní části je to kolektor a z horní části je to správa a prezentace dat. A konečně modrý pruh označuje část, kterou lze chápat jako stanici pro prezentaci výsledků a správu databáze z diagramu fyzické architektury systému.

Vidíme, že tyto vrstvy znázorňují všechny části systému stejně dobře jako předchozí diagramy, ale úhel pohledu na systém je poněkud odlišný. Vrstvy si popíšeme v tom pořadí, v jakém jsou data monitorována, zpracována a přenesena na kolektor, aby byla zřetelná činnost jednotlivých vrstev a současně systému jako celku.

1. **Z pohledu agenta:**
   - **Monitorovací vrstva** – jejím úkolem je sledovat veškerou činnost prováděnou na stanici, kde je agent nainstalován. Sleduje činnost procesů, nadpisy aktivních oken, časy, kdy jsou příslušná okna aktivní, popř. kdy byla okna zrušena atd. Stejně tak hlídá, kdy se který pracovník na stanici přihlásil, odhlásil, a jak dlouho byl aktivní v různých aplikacích. Vše provádí tak, že vytváří objekty, které interpretují příslušnou událost. Pokud např. pracovník na stanici spustí aplikaci, monitorovací vrstva zjistí vznik nového procesu a vytvoří objekt (viz str. 35), který v sobě ponese informace o této události. Jakmile je objekt vytvořen, předává řízení nižší vrstvě.

   - **Serializační vrstva** – úkolem této vrstvy je zápis vznikajících objektů (a tedy vysledovaných dat) do souboru. Objekt by mohl být zapsán do souboru v textové podobě, ovšem tento přístup má celou řadu nevýhod. Mezi hlavní nevýhody patří zejména značná složitost při zpětné rekonstrukci těchto objektů na straně kolektoru, snadná čitelnost souboru člověkem a s tím související nízké zabezpečení citlivých dat, a v neposlední řadě větší velikost souboru. Z těchto důvodů se objekty zapisují v binární podobě. K tomu je nutné objekty nejprve serializovat, což je proces, který vytvoří z objektu sekvenci bajtů, kterou pak lze do souboru zapsat.

- **Komprimační vrstva** – množství dat, které bude přenášeno na kolektor, záleží na dvou faktorech. Prvním je perioda, se kterou budou data stahována ze stanic a druhým faktorem je činnost a vytížení stanic (de facto jak moc budou pracovníci přepínat mezi okny). Z tohoto důvodu zavádíme komprimační vrstvu, jejímž úkolem bude připravit data pro přenos tím způsobem, že zmenší velikost přenášených dat pomocí některé z komprimačních metod. Díky této vrstvě se značně zrychlí přenos dat a tím se omezí i možné chyby během přenosu. Komprimace samotná je prováděna funkcí „*GzipStream*“, která, jak již název napovídá, komprimuje metodou GZIP.

- **Vrstva síťové komunikace** – úkolem této vrstvy je zajištění spojení a přenosu komprimovaných dat na kolektor. Formát dat a komunikační protokol, který mezi sebou vrstvy používají bude probrán v kap. 5. V této vrstvě jsou před samotným přenosem převedena komprimovaná data do streamu*, a může být zahájen přenos.

2. **Z pohledu kolektoru:**
   - **Vrstva síťové komunikace, komprimační vrstva, serializační vrstva** – prvními třemi vrstvami se nebudeme podrobněji zabývat, jelikož mají velmi podobnou funkci jako stejnojmenné vrstvy na straně agenta. Jediný rozdíl je, že komprimační vrstva nyní pochopitelně provádí dekomprimaci a serializační vrstva je odpovědná ze deserializaci.

   - **Databázová vrstva** – tuto vrstvu lze popsat jako rozhraní mezi kolektorem a databází. Veškerá komunikace s databází je prováděna prostřednictvím této vrstvy. Od serializační vrstvy přijímá restaurované objekty, které musí inteligentním způsobem zpracovat a na základě nich vytvořit záznamy v databázi. Tato část je implementována automatem, který na základě přijatého objektu provádí příslušné kroky popř. mění svůj stav (viz str. 50).

Předchozí vrstvy popisují část označenou jako kolektor. Než budeme pokračovat v popisu poslední části systému sloužící pro prezentaci výsledků a správu databáze (budeme souhrnně nazývat prezentační část), musíme si pro přehlednost povědět více informací o databázové vrstvě. Ve skutečnosti je totiž popis na obr. 5 zjednodušený v tom smyslu, že databázová vrstva kolektoru a

* Překlad tohoto slova je proud či tok. Tento termín je však již natolik užíván, že u něj zůstaneme. Asi nejlépe si lze pod tímto termínem představit datový tok.

prezentační části není jedna a tatáž. Pro náš popis je však obr. 5 naprosto dostačující. Nyní následuje popis poslední části systému, tj. prezentační části.

3. **Z pohledu prezentace:**
   - **Databázová vrstva** – tato vrstva slouží pro veškerou komunikaci s databází ze strany prezentační části. Od prezentační vrstvy přijímá informace, které dále zpracovává. Těmito informacemi, mohou být např. dvě data (ve smyslu datum a čas) a název aplikace. Na základě těchto informací databázová vrstva sestaví dotaz SQL, ve kterém budou figurovat právě tyto informace. Databázový systém dotaz zpracuje a jeho výsledek předá databázové vrstvě, která jej předá prezentační vrstvě.

   - **Prezentační vrstva** - jelikož jde o systém, který bude pracovat s velkým množstvím dat, je velmi důležité, aby tato vrstva provedla inteligentní prezentaci těchto dat. Jde v podstatě o komunikační rozhraní mezi vrstvou databázovou a aplikační, která poskytuje databázové vrstvě informace pro tvorbu dotazů a získané výsledky vhodně prezentuje pro aplikační vrstvu. Její činností může být např. tvorba dotazů SQL, zpracování GET parametrů apod.

   - **Aplikační vrstva** – na nejvyšší úrovni dochází již ke komunikaci mezi strojem a člověkem. Součástí této vrstvy je vhodné uživatelské rozhraní, které poskytuje formuláře, zobrazuje informace a zpracovává všechny interakce s uživatelem (v našem případě s administrátorem). Pomocí této vrstvy může administrátor volit operace, které chce s daty provádět, jako je jejich třídění, vyhledávání, mazání atp.

# 5 Formát, přenos a bezpečnost dat

Doposud jsme se věnovali především požadavkům, návrhům a modelům systému. Od této kapitoly se budeme věnovat tomu, jak splnit požadavky kladené na systém, a jak implementovat části systému tak, aby korespondovaly s vytvořenými modely. Při popisu databázové vrstvy na straně kolektoru (viz str. 30) jsme se zmínili, že tato vrstva je založena na principu konečného automatu. Aby mohl tento automat správně fungovat, musí dostávat data v očekávaném pořadí. Z toho vyplývá, že musí existovat pravidla určující formát těchto dat. Následující podkapitoly se budou věnovat komunikaci, která probíhá mezi agentem a kolektorem a ukážeme si, jak vypadají data, která se mezi nimi přenáší. Na závěr této kapitoly se budeme věnovat otázce citlivosti dat a zabezpečení, které je možné poskytnout pro jejich bezpečnost ať už na straně agenta, kolektoru, anebo při přenosu mezi nimi.

## 5.1 Data a jejich formát

Jak víme z předchozích kapitol, agent monitoruje stanici a ukládá data do souboru. Hlídá probíhající události na stanici, a při zjištění nové události vytvoří objekt, který to serializuje a uloží. Formát dat má dva důležité faktory. Prvním je fakt, že musí mít přesně dané pořadí objektů v tomto souboru, a to z důvodu zpracování kolektorem. A druhým faktorem je, že pokud je to možné, neměly by žádné atributy objektu zůstat prázdné. Tento faktor je důležitý z důvodu udržení konzistence v databázi. Popíšeme si jednak tyto objekty i pořadí, v jakém jsou ukládány do souboru. Toto pořadí si popíšeme pomocí obr. 6, který znázorňuje formát souboru, který bude přenášen. Musíme si však uvědomit, že vzhled reálného souboru nelze nikdy přesně zachytit, protože nemůžeme vědět, kdy dojde k jeho přenosu na kolektor. Z tohoto důvodu na obrázku vidíme, že je nakreslen tak, jakoby pokračoval do nekonečna. Situace, při které vznikne formát popsaný na obrázku, je taková, že na straně agenta neexistuje žádný soubor a agent byl právě nastartován.


***Obr. 6*** *– Formát přenášeného souboru*

Obrázek by nám mohl připomínat paket síťové komunikace, ovšem s paketem jako takovým nemá nic společného, pouze jsme použili podobný popis. Nyní si popišme jednotlivé části, které obrázek znázorňuje. Význam jednotlivých částí si vysvětlíme jak z pohledu agenta, tak z pohledu kolektoru, proto se jedná o syntaktický, ale i o sémantický popis formátu dat:

- **FH – File Header** – jedná se o hlavičku souboru. Pokud agent započal svoji činnost a zjistil libovolnou událost, chystá se ji zaznamenat do souboru. Tento soubor otevře a pokud zjistí, že se nachází na začátku tohoto souboru, jinak řečeno, že soubor je prázdný, musí nejprve soubor označit hlavičkou souboru. To provede serializací objektu „*FileHeader*“, který obsahuje atributy „*agentVersion*“ a „*dataVersion*“, které naplní v konstruktoru objektu. Verze agenta zde má pouze orientační charakter, který se vypisuje pouze do reportu činností kolektoru a agenta (viz str. 42).

- **LP – Login Person** – jde o záznam o přihlášeném uživateli. Díky tomu, že agent běží jako služba, tak je na stanici v provozu i ve chvíli, kdy není nikdo přihlášen. Proto je pro něj snadné zaznamenat událost přihlášení pracovníka. Pokud k této události dojde, vytvoří objekt „*LoginPerson*“, kde do atributu „*userDomainLogin*“ zaznamená login pracovníka i s doménou, a v atributu „*sessionStart*“ zaznamená čas tohoto přihlášení. Zvláštní u této události je, že ačkoliv se objekt nazývá „*LoginPerson*“, je používán při jakékoliv změně přihlášení. Tento objekt je proto využit i ve chvíli, kdy se pracovník odhlašuje. Na straně kolektoru tento objekt hraje důležitou roli, protože díky atributu „*userDomainLogin*“ zjišťuje, o kterého pracovníka se jedná a pak jsou všechny další události, zachycené na stanici, ukládány k tomuto pracovníkovi. Informace obsažené v atributu „*sessionStar*t“ jsou opět ukládány pouze do reportu. Tento objekt je zaznamenán rovněž ve chvíli, kdy agent zjistil, že už zapisuje do nového souboru, proto se v souboru nachází vždy na druhé pozici.

- **AS – Agent Started** – po startu agenta je nutné zaznamenat událost, že byl agent spuštěn. K tomuto účelu slouží objekt „*AgentStarted*“ s atributy „*agentVersion*“, což je verze agenta a „*agentStart*“, tedy čas, kdy byl agent spuštěn. Kolektor zapisuje, jaká verze agenta byla na právě zpracovávané stanici naposledy detekována, a kdy k tomu došlo.

- **DATA** – touto položkou popisujeme v souboru data, u kterých nelze přesně popsat jejich pořadí. Popíšeme si však následující objekty, ze kterých se data skládají:

- **Process Started** - jakmile na stanici vznikne nový proces, agent tuto událost zjistí a zaznamená ji vytvořením objektu „*ProcessStarted*“. Tento objekt má nejvyšší počet atributů, které si nyní popíšeme:
  - „*processId*“ - tento atribut zaznamenává tzv. identifikátor procesu známý pod zkratkou PID (process identification). Tato hodnota nijak nesouvisí s identifikátorem uloženým v databázi, tj. nesouvisí s primárním klíčem.
  - „*processPathName*“ - atribut popisující cestu k procesu. Zpravidla jde o cestu k exe souboru, kterým je proces spuštěn.
  - „*processFileVersion*“ - atribut popisující verzi exe souboru, který proces spustil.
  - „*processLegalCopyright*“ - tato informace se na straně kolektoru neukládá do databáze, pouze se uvádí do reportu. Jde o informaci, která nám říká, kdo vyrobil soubor, který proces spustil. Popř. jaké jsou možnosti jeho šíření.
  - „*processProductName*“ - v tomto atributu je uveden název produktu, ke kterému se váže spustitelný soubor, který proces vytvořil.
  - „*processProductVersion*“ - zde je uvedena verze produktu souboru.
  - „*processCompanyName*“ - tento atribut udržuje informaci o společnosti, která vyrobila program, a tedy i soubor, který proces spustil.
  - „*processOriginalFileName*“ - v tomto atributu je originální název souboru, který proces spustil. Tento název může být odlišný od názvu souboru v atributu „*processPathName*“. Jinak řečeno „*processPathName*“ obsahuje cestu s názvem tohoto souboru, který však může být přejmenován, zatímco „*processOriginalFileName*“ udržuje název, který je neměnný.
  - „*processStart*“ - je posledním atributem a je v něm obsažen časový údaj o začátku procesu.

- **Process Terminated** - podobně jako vznik nového procesu zaznamená agent i událost, že proces byl právě ukončen. K tomuto účelu slouží objekt „*ProcessTerminated*“, kde agent vyplní atribut „*processId*“ sloužící jako jedinečný identifikátor procesu a atribut „*processEnd*“ nesoucí informaci o čase, kdy proces zanikl. Atribut „*processId*“ nese klíčovou hodnotu, pomocí které kolektor najde záznam o tomto procesu v databázi. Tento identifikátor je však unikátní pouze na dané stanici mezi běžícími procesy (tj. nesouvisí s primárním klíčem v databázi). Proto atribut „*processId*“ není jediným prvkem, pomocí kterého kolektor záznam procesu v databázi vyhledává.

- **Window Activated** - v operačním systému Windows XP je vždy nějaké okno, které je tzv. aktivní. Jen připomeňme, že aktivitou okna máme na mysli událost, kdy je okno nějakého programu na popředí. Za změnu okna samozřejmě považujeme i změnu nadpisu okna (jde zejména o případ tzv. surfování po internetových stránkách), i když se prakticky žádné okno nemění. I když nemáme spuštěn žádný program s uživatelským rozhraním, systém považuje za aktivní okno plochu, popř. spořič obrazovky. V případě zjištění události, kdy se stalo nějaké okno aktivní, agent vytvoří objekt „*WindowActivated*“, který obsahuje atributy „*windowTitle*“, kde se ukládá titulek (nadpis) okna, „*processId*“, kde se ukládá identifikátor procesu, který dal oknu vzniknout a konečně atribut „*windowActivationTime*“, kde se ukládá čas, kdy se stalo toto okno aktivním. Kolektor tento objekt využívá jednak pro zaznamenání okna mezi aktivní, ale také pro zaznamenání času, kdy se předchozí okno stalo neaktivním.

- **Person Activity** – tento objekt bude v souboru vždy následovat za objektem „*Window Activated*“, protože v sobě nese informace o pracovníkově neaktivitě v předchozím okně. Kolektor pak při zpracování dat ví, kde má tuto informaci hledat. V aktivním okně se poměrně spletitým algoritmem (více v kap. 6.1.1) sleduje, zda je pracovník aktivní. Ve chvíli, kdy dojde k tomu, že se stane aktivním jiné okno, je nutno zaznamenat, jak dlouho byl uživatel aktivní v okně předcházejícím. K tomuto účelu agent vytváří objekt „*PersonActivity*“, kde zapíše zjištěné údaje do atributu „*personInactivityTime*“. Jak název napovídá, tak se jedná ve skutečnosti o to, že se zapisuje, jak dlouho byl pracovník v okně neaktivní (viz str. 45).

- **AT – Agent Terminated** – jakmile dojde k ukončení agenta, tak samozřejmě přestává monitorovat. Proto jsme tuto položku znázornili v obrázku až za data samotná. Při ukončení agenta je vytvořen objekt „A*gentTerminated*“, který má jediný atribut s názvem „*agentEnd*“, který slouží k zaznamenání času ukončení agenta. Objekt „A*gentTerminated*“ není zaznamenáván do databáze, ale pouze do reportu. Slouží proto pouze pro informaci v případě problémů (viz str. 42).

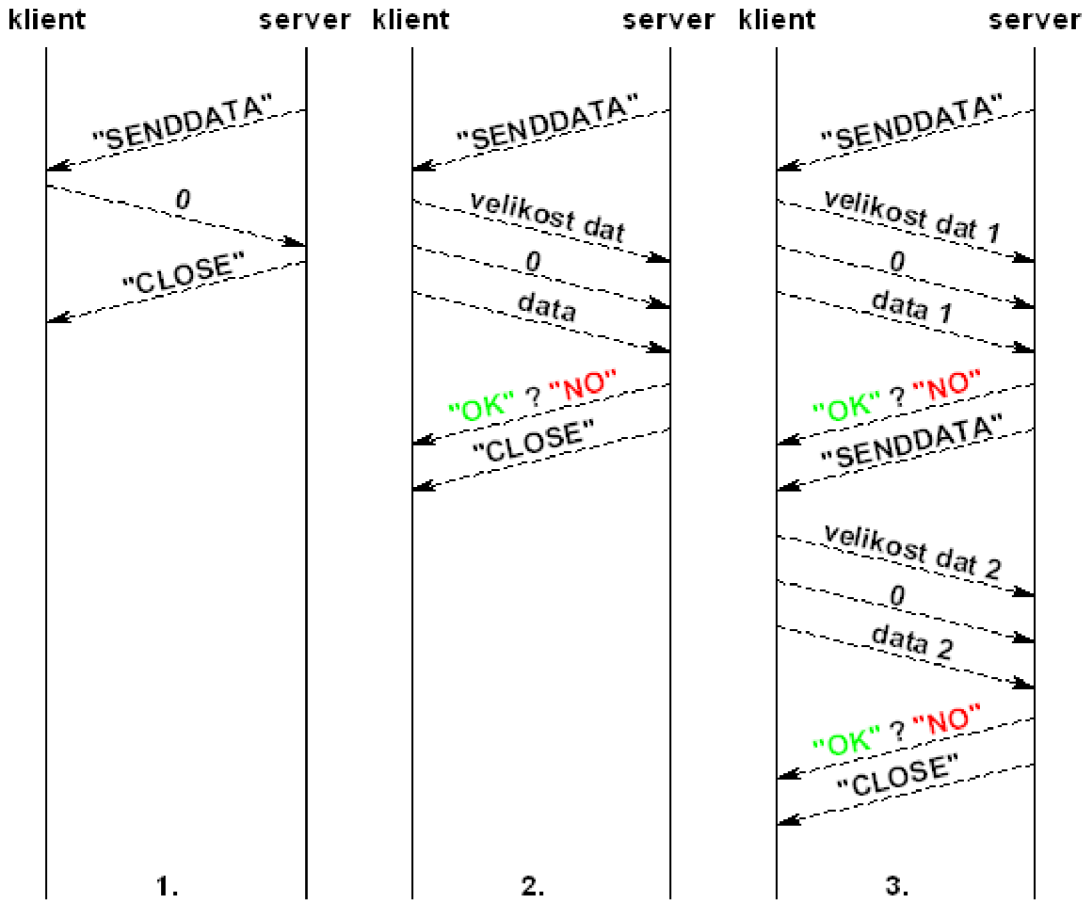
Závěrem této podkapitoly bychom rádi upozornili na fakt, že přestože jsou ukládány skutečně jen ty nejnutnější informace, výsledný soubor je poměrně obsáhlý. Mohlo by se zdát, že je zbytečné ukládat informace, které nejsou na straně kolektoru zapisovány do databáze, ale jde vždy o informace, které se vyskytují v daném souboru pouze na jednom či dvou místech a jako takové jsou z pohledu velikosti souboru zanedbatelné. Hlavním důvodem neobvyklé velikosti souboru je serializace, která je prováděna nepříliš efektivním způsobem. Serializace by šla zřejmě vylepšit překrytím nějaké bázové třídy, která je za serializaci odpovědná a snížit tak „upovídanost“ použité metody. Prozatím však budeme pracovat se současnou serializací a necháme tyto kroky až jako případná rozšíření a vylepšení projektu.

## 5.2 Komunikace agent-kolektor a komunikační protokol

Už víme, z čeho se přenášená data skládají, jaké je jejich pořadí, tj. formát těchto dat, a nyní se můžeme věnovat jejich přenosu na kolektor. Nejprve si ale povíme, jak zajistíme data proti tomu, aby se neztratila v případě předčasného ukončení agenta nebo přerušení spojení. Ještě si řekněme, že budeme používat při popisu komunikace zavedené termíny klient (tj. agent) a server (tj. kolektor).

Klient pracuje tak, že ukládá události, tj. zapisuje objekty do souboru. Ve chvíli, kdy server naváže spojení, zasláním žádosti „SENDDATA“, tak klient soubor, do kterého ukládá, uzavře, přejmenuje a začne jej odesílat. Pro ukládání událostí mezi tím založí nový soubor. Tím pádem dochází k přenosu pouze tohoto přejmenovaného souboru. Během přenosu však může dojít k nějaké chybě. Data však nemohou být ztracena, protože zůstanou ve starém souboru na straně klienta a smazána jsou teprve až když proběhne přenos bez problémů. Pokud k této chybě dojde, při dalším navázání spojení nastává případ, kdy klient musí poslat soubory dva. Průběh je takový, že server pošle žádost o data, klient chce aktuální data uložit a přejmenovat, ale zjistí, že existuje ještě starý soubor. Proto klient pošle indikátor, pomocí kterého serveru oznámí, že se budou posílat soubory dva

a prvním posílaným souborem je ten starší. Tím by měl být vyřešen problém se ztrátou dat během přenosu i celá komunikace. Tento komunikační protokol je znázorněn na obr. 7. Jak celý přenos probíhá z technického hlediska si vysvětlíme v kapitole 6.1.2.




***Obr. 7** – Komunikace mezi klientem (agent) a serverem (kolektor)*

Tento obrázek popisuje tři komunikace, ke kterým může během přenosu dojít:

1. Každá komunikace začne vždy tím, že server pošle zprávu „SENDDATA“. Klient tuto zprávu přijme a podívá se, zda má nějaký soubor, který by mohl odeslat. Server očekává jako odpověď velikost dat, která mu budou posílána. Pokud však klient zašle hodnotu nula, tak jako to vidíme na obrázku, tak server ví, že klient žádný soubor nemá, a tudíž se žádný přenos konat nebude. Server tedy ukončí spojení zprávou „CLOSE“.

2. V druhém případě již klient má co nabídnout. Server jej osloví zprávou „SENDDATA“ a odpovědí mu je velikost dat, která budou přenášena. Tuto velikost si server vhodně uloží, aby později mohl zkontrolovat, zda odpovídá velikost přijatého souboru. Další, co přijme je indikátor toho, zda bude přenášen jeden nebo budou přenášeny dva soubory. V tomto případě přijme nulu, což znamená, že přenášený soubor bude pouze jeden. Potom následuje přenos samotných dat, a po jejichž přijmutí server kontroluje zmíněnou velikost. Potom server potvrzuje přijetí řetězcem „OK“, anebo posílá řetězec „NO“, pokud došlo k nějakému problému. Jakmile server přijme všechna data, ukončí spojení zprávou „CLOSE“. V případě, že klient přijal řetězec „NO“, tak je komunikace ihned ukončena a nedochází již k poslání řetězce „CLOSE“.

3. Třetí a poslední verze popisuje případ, kdy má klient ještě starý nepřenesený soubor. Komunikace je navázána opět zprávou „SENDDATA“ ze strany serveru a odpovědí mu je znovu velikost dat. Server si opět uloží velikost dat a přijme již zmíněný indikátor. V tomto případě přijme server jedničku, což znamená, že si po přijmutí souboru má říct o další data. Klient začne posílat data a server je opět kontroluje s velikostí, kterou přijal na začátku. Jakmile server zjistí, že soubor dorazil celý, opět požádá o data zprávou „SENDDATA“. Klient opět pošle velikost dat, které bude posléze posílat. Aby bylo dodrženo správné pořadí a tím pravidla protokolu, tak dále klient pošle indikátor, což je tentokrát nula. Potom jsou opět posílána data a server je přijímá, dokud nejsou přijata celá. Jakmile server zjistí, že velikost přijatých dat se rovná velikosti dat dopředu oznámených a zjistí také, že poslaný indikátor byl nula tak opět zašle potvrzení „OK“ nebo „NO“ a popř. zprávu „CLOSE“.

Samozřejmě může dojít i k jiné neočekávané chybě (např. vytažení síťového kabelu), kdy se přeruší komunikace dříve, než je schopen server potvrdit stav přijetí. V takovém případě dojde k výjimce, ale data se opět nemohou ztratit. Jednoduše řečeno, klient smaže soubor pouze a jenom pokud přijme řetězec „OK“. Řetězce „OK“ a „NO“ nejsou zvoleny náhodou, ale musí mít z důvodu snazší implementace stejný počet znaků. V případě přijetí řetězce „NO“ je komunikace mezi serverem a klientem okamžitě ukončena.

Komunikace v našem systému je implementována pomocí protokolu TCP. Jedná se o protokol zajišťující spolehlivý přenos dat. Bohužel kromě výhod má pochopitelně i své nevýhody. TCP totiž dosahuje své spolehlivosti neustálou kontrolou přenášených dat. Proto má mnoho řídících informací,

je poměrně pomalý a přenos dat tím pádem ztrácí na efektivitě. Naproti tomu existuje protokol UDP, který nezaručuje spolehlivost přenosu, ale je mnohem rychlejší, protože má menší hlavičku a neposílá potvrzování [4].Důvodem, proč byl zvolen protokol TCP, je samozřejmě jeho spolehlivost. Nejde ani tak o ztrátu dat jako takových, ale hlavně o to, že pokud má kolektor data inteligentně zpracovat, tak existují sekvence dat, které musí jít přesně za sebou. I pokud by se kolektor vypořádal s problémy plynoucími z použití UDP protokolu, nejspíš by v databázi vznikaly datové anomálie, které by měly značný vliv na výslednou prezentaci vysledovaných výsledků. Nevýhodou naproti tomu je, že síť může být tímto přenosem značně zatížena. Prozatím však implementaci přenosu dat ponecháme s protokolem TCP. Případné změny by mohly být otázkou možných rozšíření systému.

## 5.3 Bezpečnost dat a bezpečný přenos

V kapitole věnující se etickým a právním otázkám jsme nastínili fakt, že vysledovaná data mohou být značně citlivá, a jako s takovými s nimi musíme nakládat. Z tohoto důvodu je nutné věnovat se otázce zabezpečení dat i zabezpečení samotného přenosu agent-kolektor. Na jedné stanici může pracovat více uživatelů. V případě špatného zabezpečení dat by mohl každý pracovník zjistit, co prováděli jeho kolegové, kteří stanici používali před ním. Musíme se proto snažit zamezit přístupu k těmto datům a to ještě před přenosem na kolektor.

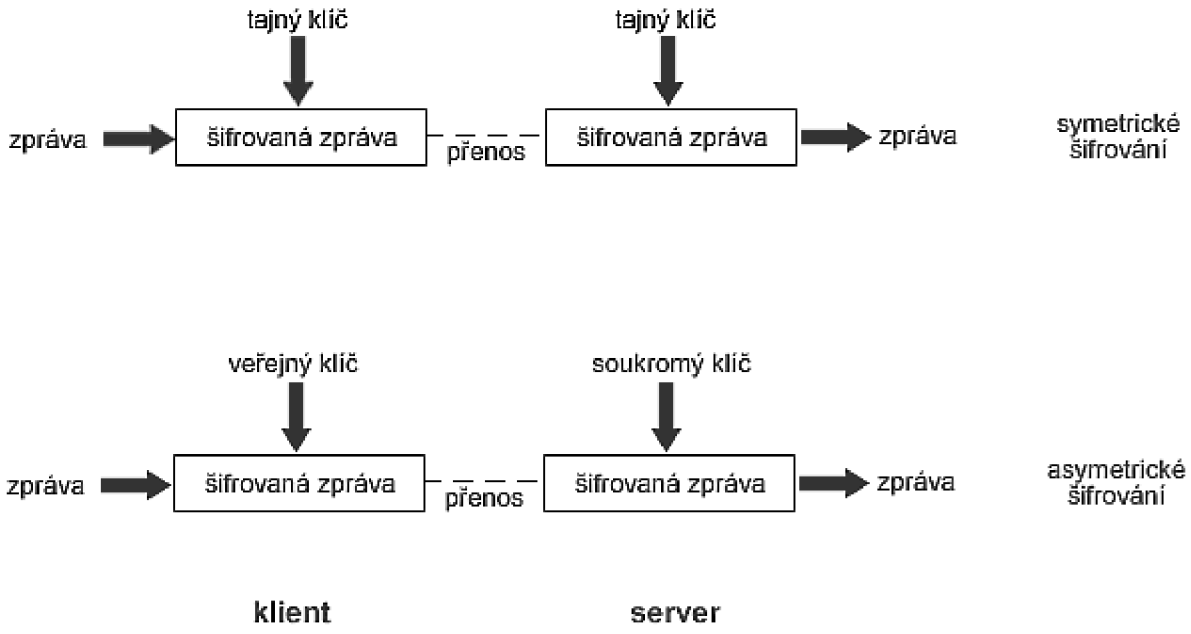
Bavíme-li se o ochraně naskenovaných dat na stanici, jde vlastně o ochranu souboru, do kterého jsou ukládány objekty symbolizující vysledované události. Proto nejlepší navrhovanou ochranou je zabezpečení tohoto souboru na úrovni operačního systému. Administrátor počítačové sítě může jednoduše zakázat uživatelům přístup buď k souboru, nebo k části adresářové struktury, anebo rovnou k celému logickému disku. Tím by byla zcela vyřešena otázka ochrany dat na stanici. Dále je třeba říci, že díky serializaci objektů je soubor velmi těžko čitelný a vyznat se v něm není lehký úkol. Tento fakt ovšem nemůže být pokládán za dostatečný pro zabezpečení citlivých informací.

### 5.3.1 Možnosti bezpečného přenosu klient-server

Otázku ochrany dat na stanici můžeme považovat za snadno řešitelnou. Otázka přenosu těchto dat na kolektor je mnohem problematičtější, a tak se pokusíme vysvětlit možnosti, výhody a nevýhody jednotlivých přístupů. Bezpečný přenos bude pouze navržen, nikoliv implementován. Důvodem je, že při vývoji systému jakožto produktu nebyl bezpečný přenos vyžadován.

Nejprve si něco povíme o symetrickém a asymetrickém šifrování ve spojitosti s naším projektem. Na obr. 8 si můžeme všimnout, že symetrické šifrování využívá pouze a jenom jeden klíč pro odesílatele i příjemce. Pokud užijeme symetrické šifrování, dosáhneme dobrých výsledků, z pohledu rychlosti šifrování.

V našem případě však musíme počítat s tím, že přenášený soubor bude poměrně velký, a proto by se nám rychlost šifrování u použití symetrického algoritmu zamlouvala. Problémem však je, že poměrně velký může být i počet stanic, ze kterých budeme tyto soubory stahovat. A zde nastává komplikace s distribucí těchto klíčů. V tomto směru by se řešením mohl stát asymetrický algoritmus, který používá dva různé klíče pro šifrování a dešifrování zprávy. Jeho značnou nevýhodou je však pomalé šifrování. V tom případě je ale řešením kombinace těchto dvou algoritmů, přesně jak to provedl Phil Zimmerman ve svém projektu PGP (viz str. 14) [3].




***Obr. 8** – Symetrické a asymetrické šifrování*

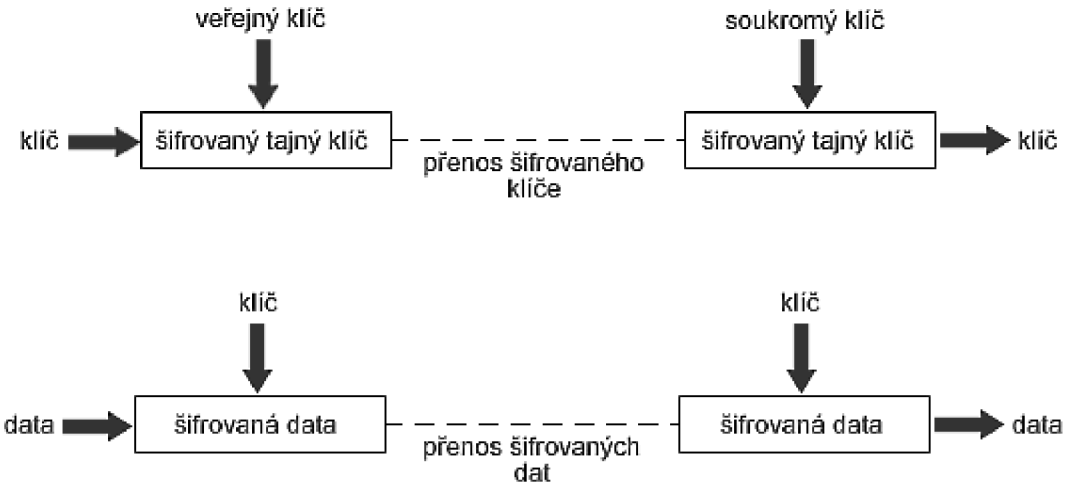
Pro řešení našeho problému je třeba, aby klient byl opatřen mechanismem pro generování klíčů pro symetrické šifrování. Dále potřebujeme, aby jeho součástí byla znalost veřejného klíče, přičemž

soukromý klíč bude součástí serveru. Před samotným přenosem souboru nejprve agent vygeneruje klíč pro symetrické šifrování, který zašifruje veřejným klíčem. Tento zašifrovaný klíč zašle na server, který ho pomocí soukromého klíče dešifruje.

Tímto krokem se vyřeší otázka distribuce klíče pro symetrické šifrování a současně i rychlost šifrování, protože asymetrickým algoritmem se šifruje pouze malé množství dat, tj. klíč pro symetrické šifrování. Samotný soubor je pak zašifrován symetrickým algoritmem, který je rychlejší a problém s distribucí klíčů nemůže nastat. Celý popsaný proces je zobrazen na obr. 9. I když vidíme, že použití námi navrženého řešení se provádí ve dvou krocích, tak výsledek by byl efektivnější, než při volbě jedné z metod z obr. 8.

Další možností a zřejmě nejjednodušší by bylo použití SSL (secure sockets layer). Jedná se o protokol, který umožňuje bezpečnou komunikaci. Běží nad TCP/IP vrstvou a doslova jde o použití bezpečných soketů. Problémem ovšem je, že podpora v .NET pro SSL pouze ve formě https, což je pro náš projekt bohužel nedostačující.

.




***Obr. 9** – Navržené řešení bezpečného přenosu*

# 6 Implementace systému

V této kapitole se budeme věnovat implementaci nejdůležitějších částí systému. Podrobně si popíšeme, jak jsou agent a kolektor implementovány, a zaměříme se na důležité programové konstrukce. Vysvětlíme si, jak jsou plněny požadavky, o kterých jsme mluvili v předchozích kapitolách, a také si povíme něco o dalších součástech systému, které k němu neodmyslitelně patří. Povíme si o možných rozšířeních a slabinách, které systém obsahuje. Jen připomínáme, že implementačním jazykem je C# na platformně .NET.

Ještě než se začneme věnovat popisu jednotlivých částí, měli bychom si říci o reportech, které agent i kolektor obsahují. Systém Windows XP umožňuje využít zaznamenání činnosti programu pomocí zapisování zpráv, které si potom můžeme prohlížet ve správě počítače pomocí prohlížeče událostí. Jde o ladící výpisy, které obsahuje jak agent, tak i kolektor, a jde v podstatě o jedinou možnost, jak v případě chyby zjistit, kde a proč k chybě došlo.

## 6.1 Implementace agenta

Je dobré si uvědomit, jak široký rozsah má náš vyvíjený systém. Samotný agent musí provádět monitorování událostí, zápis do souboru, přenos dat apod. Tyto činnosti musí umět vhodně zkombinovat. Jde o poměrně rozsáhlou problematiku, a proto se pokusíme zaměřit na nejdůležitější prvky, které agent obsahuje.

Již bylo řečeno, že v praxi je agent nasazován výhradně jako služba. Běží tedy na pozadí systému a uživatel stanice nemá možnost si všimnout jeho přítomnosti na stanici. Pro účely vývoje je však možné jej spustit i jako aplikaci s uživatelským rozhraním, a tak byl i vyvíjen, neboť ladění a hledání chyb ve službě je velmi náročné. O tom, jak má právě agent běžet, se rozhoduje na základě parametrů, které jsou předávány při spouštění. Agent je postaven na principu běhu dvou vláken. První vlákno, kterému budeme říkat vlákno monitorovací, je odpovědné za monitorování uživatelských aktivit a zápis zjištěných událostí (objektů) do souboru „*DataCurrent.bin*“. Vlákno druhé, které budeme nazývat vlákno komunikační, musí sledovat, zda nepřichází žádost o síťové spojení. V případě, že dojde k žádosti o spojení a přenos dat, musí toto vlákno řídit přenos souboru na kolektor. Jelikož jedno vlákno do souboru zapisuje události a druhé se může snažit přenášet tento soubor na

kolektor, musí obě vlákna používat systém vzájemného vyloučení, aby měla jistotu, že pokud k souboru přistupují, tak s ním právě nepracuje druhé vlákno.

Hlavní metoda, ze které jsou tato dvě vlákna spuštěna, nejprve zkontroluje, zda existuje soubor „*DataCurrent.bin*“, a překontroluje verzi dat tohoto souboru. Pokud verze neodpovídá aktuální verzi agenta, tak soubor jednoduše odstraní. Poté teprve spustí monitorovací a komunikační vlákno a dále pouze čeká v nekonečné smyčce, dokud nepřijde událost ukončení agenta. Po celou dobu pouze přepíná kontext, aby mohla pracovat zmíněná vlákna. Pokud je agent ukončen, tak tato metoda ukončí vlákno monitorovací, tj. zastaví časovač, kterým je vlákno řízeno a zapíše do souboru poslední událost, ukončení agenta.

### 6.1.1 Vlákno monitorovací

Vlákno odpovědné za monitorování běží v nekonečném cyklu, jehož interval průchodu je řízen časovačem, a díky tomu může v udaném intervalu monitorovat. Po vstupu do metody tohoto vlákna musíme zajistit výlučný přístup ke kódu tohoto vlákna pomocí konstrukce „*lock (this.timer)*“, protože pokud by byl interval průchodu příliš krátký, mohlo by teoreticky dojít ke zdržení v jednom průchodu, a mezitím by začal další průchod vláknem. To by mohlo způsobit značné problémy, které by mohly narušit korektní způsob monitorování.

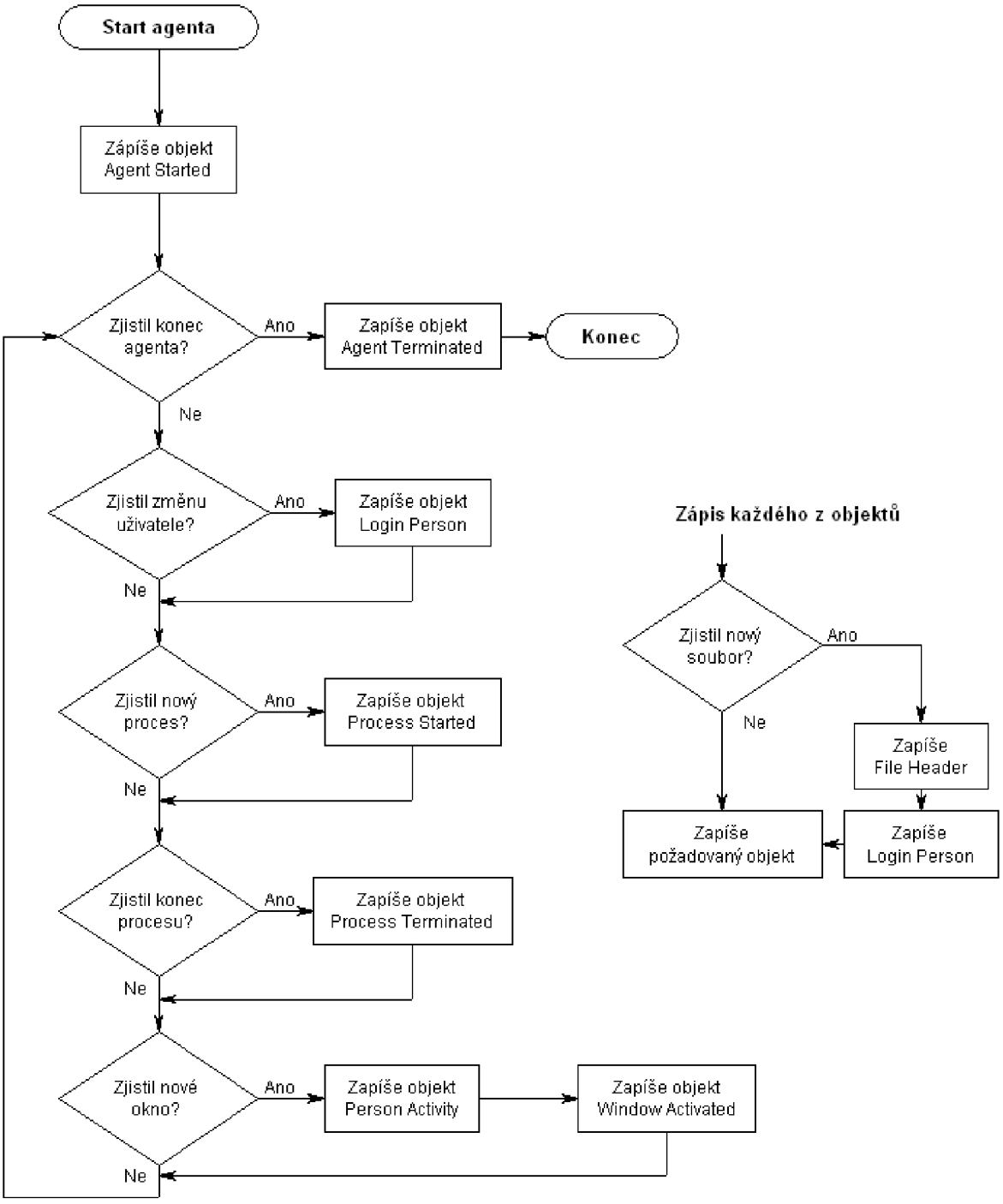
Celý algoritmus monitorování je poměrně složitý pro slovní popis. Využijeme proto obr. 10, který tento algoritmus znázorňuje a jednotlivé části si v následujících podkapitolách popíšeme více do hloubky.

#### Zápis událostí

V pravé části obrázku vidíme menší diagram nadepsaný „Zápis každého z objektů“. Tento obrázek popisuje princip, na jakém dochází k zápisu všech objektů. V podstatě popisuje tělo metody, která je volána vždy, když potřebujeme zapsat některý z objektů. Jak popisuje diagram, součástí metody je právě kontrola, zda neprovádíme tento zápis do nového souboru. V takovém případě založí soubor nový, na jehož začátek zapíše nejprve objekty „*FileHeader*“ a „*LoginPerson*“. Co ovšem z diagramu nevyčteme je, že při každém zápisu do souboru si musí zajistit k tomuto souboru výlučný přístup. Tak má jistotu, že s ním právě nebude pracovat druhé, komunikační vlákno.

#### Popis algoritmu monitorování

Větší diagram v levé části obrázku začíná zápisem objektu symbolizující start agenta. V tuto chvíli ještě monitorovací vlákno neběží, proto je tato událost zaznamenána v hlavní metodě třídy agenta.




***Obr. 10*** *– Diagram znázorňující činnost monitorovacího vlákna*

Stejně tak následující kontrola, zda nedošlo k ukončení agenta, je prováděna v hlavní metodě. Při této kontrole ale už monitorovací vlákno běží, a z hlavní metody dojde k přepnutí kontextu v případě, že k ukončení nedošlo.

Následující kroky jsou již implementovány přímo v monitorovacím vlákně. Vlákno nejprve zjistí, zda nedošlo ke změně přihlášeného uživatele. Tato událost je velmi důležitá, protože veškeré další události jsou vždy vázány k tomuto uživateli. Kolektor díky tomu pozná, ke kterému uživateli má získaná data zapisovat. Potom musíme zjistit, zda vznikly nějaké nové procesy. Tato část algoritmu je řešena na základě porovnání dvou struktur, z nichž první je globální, a tedy obsahuje procesy i z předchozího průchodu vláknem. Druhou strukturu naplníme v každém průchodu aktuálně existujícími procesy. Na základě tohoto srovnání snadno zjistíme, které procesy jsou v aktuálním průchodu vláknem nové. Na velmi podobném principu je založeno i zjišťování procesů, které byly zrušeny. Tento mechanismus však probíhá přesně naopak. Zjišťujeme tedy, zda všechny procesy, které jsou uloženy v globální struktuře, se rovněž vyskytují ve struktuře, ve které jsou uloženy aktuálně existující procesy.

Nejsložitějším krokem monitorovacího vlákna je zaznamenání neaktivity uživatele v aktuálním okně a zjištění aktuálně aktivního okna. Zaznamenávání neaktivity funguje na principu zjišťování doby posledního uživatelského vstupu. Musíme si být vědomi, že k přepnutí oken může dojít i samovolně, např. pokud se aktivuje spořič obrazovky. Díky tomu se musí započítávat nejen doba mezi jednotlivými uživatelskými vstupy, ale i doba od začátku okna, které se stalo aktivním bez uživatelské aktivity. Tyto hodnoty se ukládají do globální proměnné při každém průchodu vláknem. Tato proměnná je vynulována až tehdy, pokud v kódu, který následuje zjistíme, že se stalo aktivním jiné okno. Připomeňme si, že za změnu aktivity okna považujeme nejen změnu oken, ale i pouhou změnu nadpisu okna. Proto se zjišťování změny oken provádí nejen na základě nadpisu a na základě tzv. handlu*, což je informace, která je pro dané okno unikátní. Je to tak proto, že mohou existovat dvě okna se stejným nadpisem.

Jakmile tedy zjistíme, že došlo ke změně aktivity okna, nastává další komplikace, kterou musíme řešit. Tentokrát se jedná o to, že interval průchodu monitorovacím vláknem může být větší než doba, která se považuje za neaktivitu v okně. Jinak řečeno, uživatelská neaktivita v okně se nezačne měřit ihned, ale je volitelná. Tuto časovou asynchronizaci řešíme kontrolou rozdílu časů změny okna a aktuálního času. Pokud zjistíme, že tento rozdíl je větší než přednastavená hodnota doby, po které se má začít sledovat neaktivita, tak se tato hodnota přičte k celkové neaktivitě pro předchozí okno, tj. pro to, které se právě stalo neaktivním.

---

* Termín handle lze chápat jako ukazatel na aktivní okno a tento termín je velmi těžké vyjádřit jinými slovy. Proto se budeme držet tohoto anglického pojmu.

**Problém zaznamenání nově vzniklých procesů**

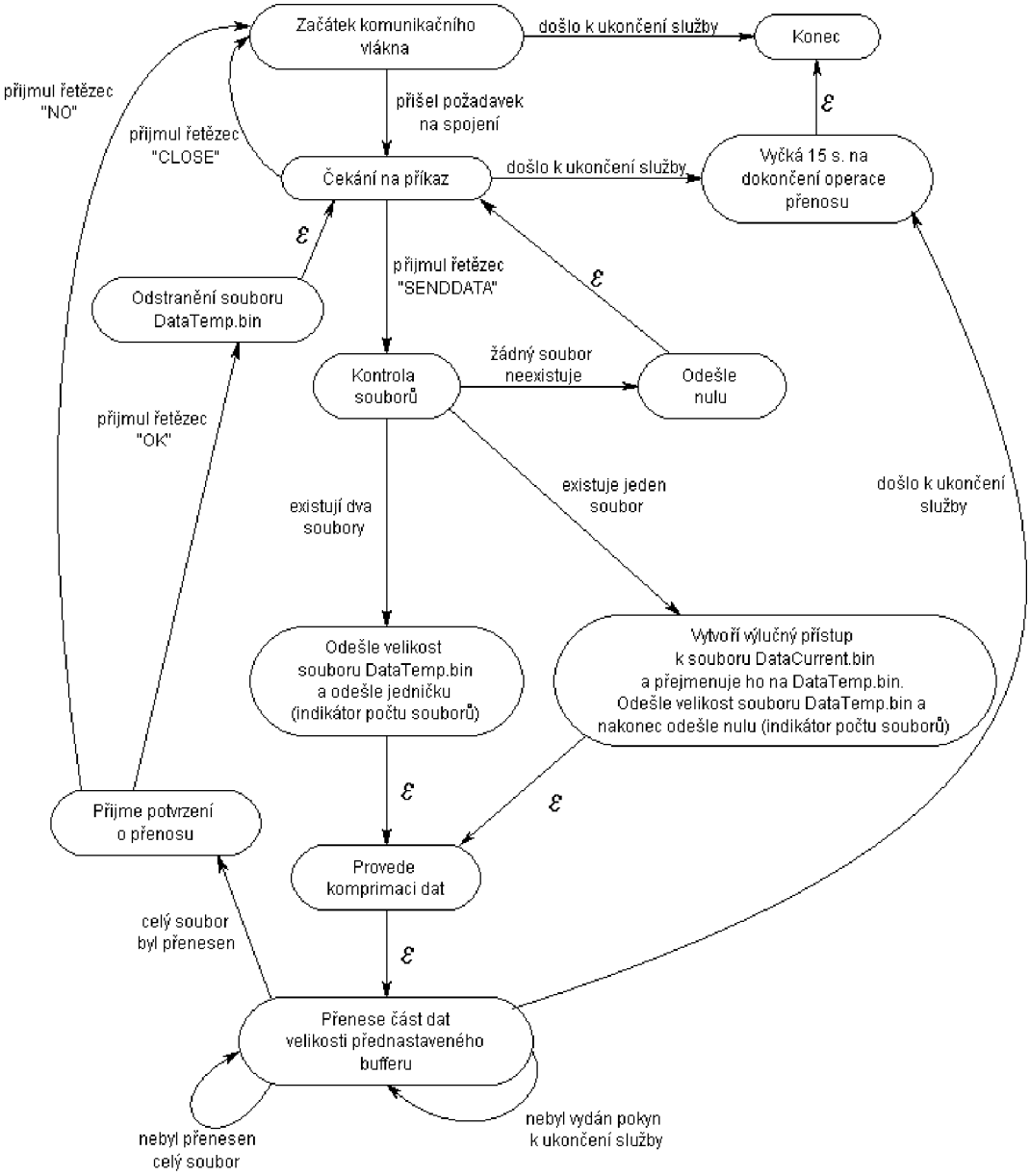
Při testování již hotového systému byla zjištěna velmi zajímavá chyba spojená se zaznamenáváním procesů. Tvorba objektu „*processStarted*“ je založena na tom, že podle PID se vytvoří systémový objekt „*Process*“, s jehož pomocí vytvoříme objekt „*FileVersionInfo*“. Ten nám dále poskytne všechny informace o procesu, které potřebujeme. Občas se však stalo, že proces, i když nešlo o žádný důležitý systémový proces apod., odmítl poskytnout přístup k těmto datům. Navíc bylo velmi obtížné tento okamžik nějak zachytit či nasimulovat.

Tento, na první pohled drobný nedostatek, však mohl mít značný vliv na výsledná data, protože docházelo k tomu, že aktivní okna neměla proces, ke kterému by se vázala (samozřejmě pouze ve výsledcích našeho systému). Nakonec bylo zjištěno, že problém je v tom, že se agent na jeho existenci dotázal příliš brzy po jeho vytvoření. Proces ještě v té době neměl vytvořeny všechny prostředky, pomocí kterých o sobě poskytoval důležité informace. Řešení bylo oproti hledání problému mnohem snazší. Dotazování na objekt „*FileVersionInfo*“ se nyní provádí ve smyčce, a pokud je přístup k tomuto objektu zamítnut, jednoduše se na okamžik přepne kontext, který umožní procesu dokončit potřebné operace.

### 6.1.2 Vlákno komunikační

Komunikační vlákno má za úkol naslouchat na příslušném portu, a v případě žádosti o spojení realizovat přenos souboru. Tuto činnost si tentokrát popíšeme s pomocí obr. 11.

Diagram, který na obrázku vidíme bychom mohli nazvat také jako stavový diagram. Stavy tohoto diagramu vyjadřují činnost provedenou vláknem. Přechody mezi těmito stavy vyjadřují události, ke kterým došlo nebo obecně pravdivá fakta (např. existují dva soubory). Některé přechody jsou označeny řeckým písmenem „*epsilon*“, což znamená, že po vykonání činnosti v příslušném stavu dojde automaticky k přechodu do dalšího stavu. Tento symbol se podobným způsobem využívá u konečných automatů, a tak jsme jej použili i v tomto diagramu. Nejprve si vysvětlíme význam stavu, který má za úkol vyčkat na dokončení operace přenosu. Jelikož agent může být ukončen při právě probíhajícím přenosu, je dobré zajistit, aby mohla být tato operace dokončena. Proto po příkazu ukončení služby agenta bude komunikační vlákno ještě nějakou dobu (v našem případě patnáct vteřin) vyčkávat, než se ukončí.




***Obr. 11*** *– Diagram popisující činnost komunikačního vlákna*

Celé komunikační vlákno je založeno na smyčce, ve které očekává buď spojení ze strany kolektoru nebo pokyn pro ukončení agenta. Jakmile dostane žádost o spojení, přejde do vnitřní smyčky, ve které je připraveno realizovat přenos souborů. Jde o stav, ve kterém čeká na příkazy z kolektoru. Obdrží-li příkaz „SENDDATA“, tak musí zkontrolovat, kolik souborů se na straně agenta nachází. V případě, že se na disku nenachází žádný soubor, pošle na kolektor nulu a vrací se zpět do

stavu, kdy čeká na další příkaz. Dalším příkazem z kolektoru by pak měl být příkaz „CLOSE“, po kterém opouštíme vnitřní smyčku. V případě, že zjistí přítomnost jediného souboru, tak si zajistí k tomuto souboru (soubor se jmenuje „*DataCurrent.bin*“) výlučný přístup, a přejmenuje ho na „*DataTemp.bin*“. Pokud zjistí přítomnost dvou souborů, tak už ví, že k přejmenování došlo při předchozím pokusu o přenos a nyní už soubor „*DataTemp.bin*“ existuje.

V dalším stavu pošle na kolektor hodnotu udávající velikost souboru „*DataTemp.bin*“. Ihned za touto velikostí odešle znak nula nebo jedna podle toho, kolik souborů se na straně agenta nachází (nula znamená jeden soubor, jednička symbolizuje dva existující soubory). Dále soubor „*DataTemp.bin*“ zkomprimuje, aby odesílaná data byla co možná nejmenší a ve smyčce začne tato data odesílat, dokud nejsou přenesena všechna nebo nebyl vydán pokyn pro ukončení agenta. A jak jsme již řekli, v případě pokynu pro ukončení agenta se pokouší ještě chvíli přenos úspěšně dokončit. Jakmile odešle soubor celý, tak přijme řetězec „OK“ nebo „NO“ (viz str. 37-38). Pokud přijmul řetězec „OK“ symbolizující úspěšný přenos, tak soubor „*DataTemp.bin*“ smaže a vrací se do stavu, kde opět čeká na příkazy z kolektoru. Takto je realizováno celé komunikační vlákno.

## 6.2 Implementace kolektoru

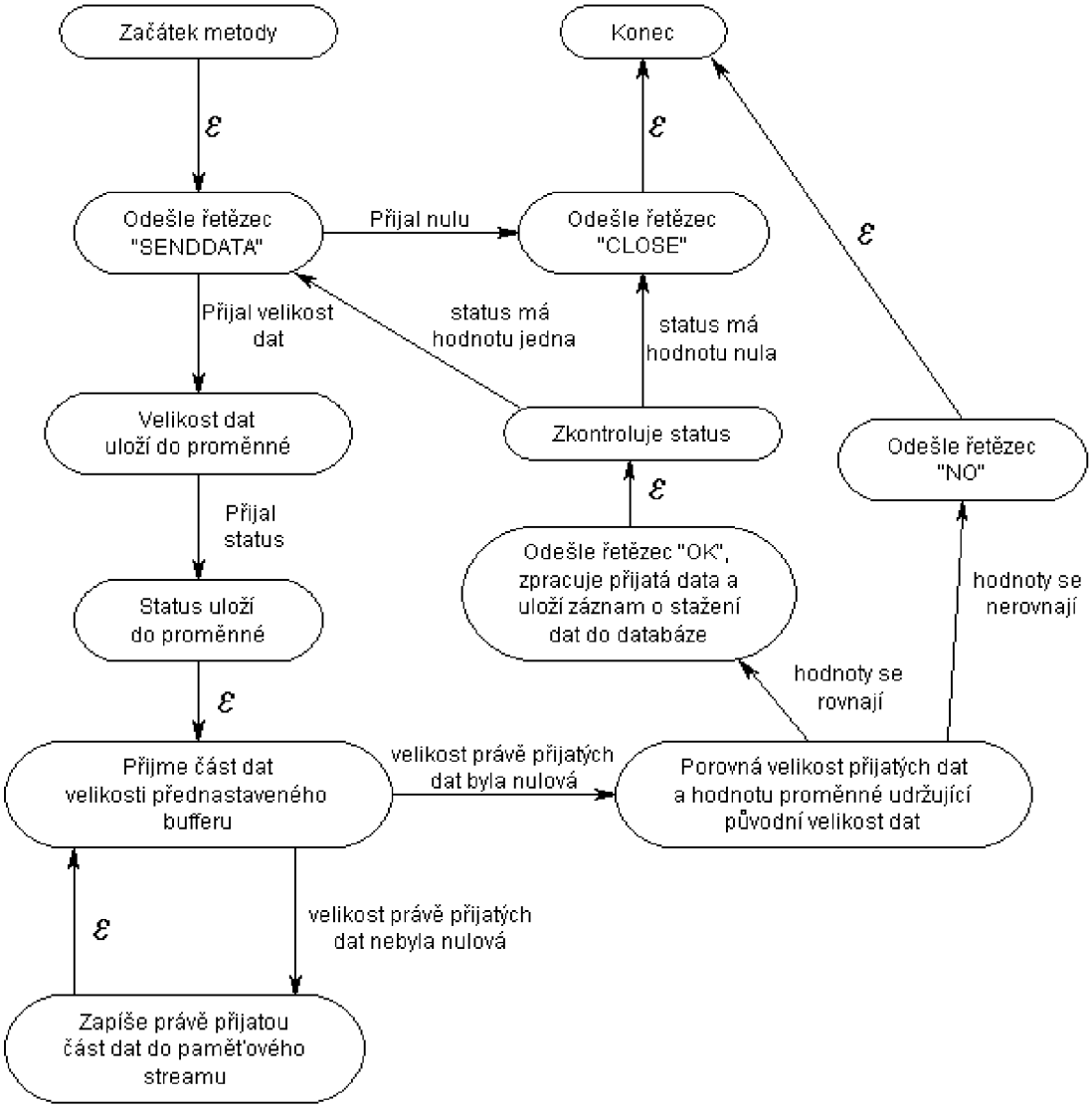
Podobně jako agent i kolektor je implementován tak, že může být spuštěn jako služba, ale i jako aplikace s uživatelským rozhraním. Princip spuštění je obdobný jako u agenta, proto se jím nebudeme dále zabývat. V praxi by měl kolektor běžet samostatně (tedy bez obsluhy člověka) jako služba. Proto je jeho základem mechanismus, pomocí kterého prochází seznam sledovaných počítačů z databáze a žádá je o data. Jak často tuto činnost vykonává, je otázka nastavení periody stahování dat.

Hlavní metoda kolektoru běží v nekonečné smyčce a podle nastavení se snaží stahovat data ze stanic. Pokud se nepodaří stáhnout všechna data, tak se po určité době (touto dobou je přednastavená doba čekání při neúspěšném pokusu), pokusí přenos opakovat.

### 6.2.1 Zpracování stanic

V metodě pro zpracování stanic se nejprve připojíme do databáze, aby mohla využít informací uložených u každé stanice, potřebných k jejímu zpracování. Takto získává dotazem SQL jména všech počítačů, které může zpracovat a to opět v nekonečném cyklu, který je podmíněn tím, že projde všechny stanice nebo je kolektor ukončen. Získané informace, tj. identifikátor a jméno stanice, potřebuje k připojení k příslušné stanici a stažení souborů.

Zpracování konkrétní stanice již řídí jiná metoda, která se ke stanici připojí, stáhne soubor a zavolá metodu, která zpracuje stažený soubor. My si tuto metodu popíšeme podrobněji pomocí diagramu na obr. 12. Diagram používá stejné prostředky, jako diagram z obr. 11. Na svém začátku metoda nastaví potřebné proměnné a připojí se do databáze. Poté odešle příkaz „SENDDATA“ na příslušnou stanici a čeká na odpověď. Tou je v každém případě číselná hodnota. V případě, že jde o nulu, zašle příkaz „CLOSE“ a ukončí svoji činnost. V opačném případě si přijatou hodnotu uloží a dále očekává status, kterým je vždy hodnota nula nebo jedna, a tuto rovněž uloží do proměnné.

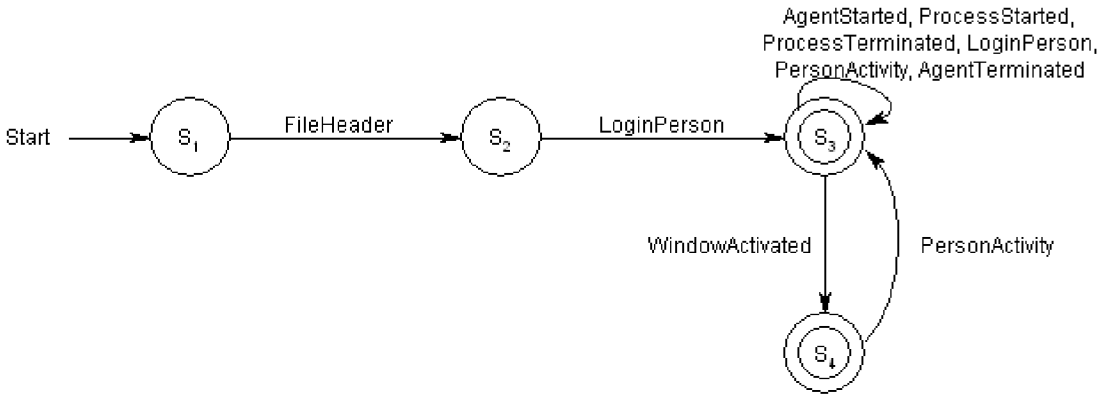



***Obr. 12** – Diagram popisující činnost metody stahující data z agenta*

Potom začne po menších blocích přijímat samotný soubor. Vždy zkontroluje, zda právě přijatá část nebyla nulová a tato data zapíše do paměťového streamu. Jakmile je hodnota velikosti právě přijatých dat nulová, musí porovnat hodnotu velikosti dat uložené v proměnné s velikostí dat vepsaných do paměťového streamu. Tyto hodnoty by se měly rovnat. Pokud se však nerovnají, zašle příkaz „NO“ a komunikace se ukončí, protože došlo k nějaké chybě přenosu. Rovnají-li se hodnoty, zašle příkaz „OK“ a pokračuje v činnosti, kterou je zpracování dat (viz. následující podkapitola). Po zpracování těchto dat kontroluje status, který si uložil. Pokud má hodnotu jedna, tak ví, že se bude posílat ještě soubor a znovu zašle příkaz „SENDDATA“. Pokud zaslaný status měl hodnotu nula, tak se žádný další soubor posílat nebude a pouze zašle příkaz „CLOSE“ a ukončí svoji činnost.

**Zpracování dat**

Zpracování samotných dat, které jsou v paměťovém streamu, provádíme další samostatnou metodou. Tato metoda čte paměťový stream a restauruje dříve serializované objekty. K popisu činnosti této metody nyní použijeme deterministický konečný automat, který můžeme vidět na obr. 13. Princip restaurace jednotlivých objektů je takový, že nad paměťovým streamem voláme metodu, která je schopna deserializovat objekt, před kterým se právě ve streamu nacházíme. Takový objekt je nejprve vytvořen jako obecný objekt, o kterém zatím nic nevíme. Snadno však zjistíme jméno objektu díky metodě, která nám zjistí typ objektu. Pak pouze vytvoříme konkrétní objekt pomocí přetypování původního obecného objektu.




***Obr. 13*** *– Konečný automat popisující zpracování dat*

Díky tomuto postupu vidíme, že množinou vstupních symbolů v našem konečném automatu jsou vlastně názvy objektů, které používáme k záznamu události vznikající při monitorování na straně

agenta. O objektech samotných se zmíníme pouze okrajově, protože jsme jim věnovali kapitolu 5.1. Navážeme však na spojitost mezi formátem dat, probíraným ve stejné kapitole, a zpracováním těchto dat v této metodě.

Jak můžeme vidět na obrázku, tak v každém případě budou první dva objekty „*FileHeader*“ a „*LoginPerson*“. Objekt, který bude následovat po těchto dvou, již prakticky nelze odhadnout. Důvodem je, že nikdy nemůžeme vědět, v jakém stavu se agent nacházel, když byl požádán kolektorem o zpracování dat. Proto jsme také u obr. 6 v kap. 5.1 zmínili, že popisuje formát dat ve chvíli, kdy je agent spuštěn a přitom není na straně agenta žádný soubor. V takovém případě je totiž třetím objektem v pořadí objekt „*AgentStarted*“. Stejně snadno ale může dojít k situaci, kdy agent běží a chystá se zaznamenat např. událost aktivity pracovníka, ale mezitím je soubor odeslán na kolektor. Pak by byl třetím objektem v pořadí objekt „*PersonActivity*“, i když víme, že musí vždy následovat bezprostředně za objektem „*WindowActivated*“. Tento objekt, tedy objekt „*WindowActivated*“, však byl zaznamenán v právě odeslaném souboru.

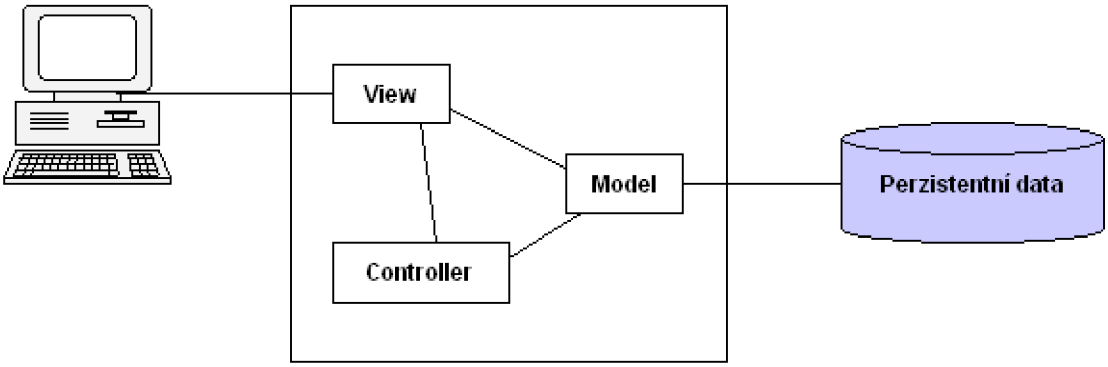
Na tyto případy musí být kolektor připraven. Proto se v automatu s objektem „*PersonActivity*“ dostáváme vždy do stavu $S_3$, a to i v případě, že se v tomto stavu již nacházíme. Fakt, že objekt „*PersonActivity*“ následuje obvykle za objektem „*WindowActivated*“ je vyjádřen smyčkou mezi stavy $S_3$ a $S_4$. Ve skutečnosti je implementovaný automat ještě o něco jednodušší než ten na obrázku, protože neřešíme návaznost těchto dvou objektů. Oba tyto stavy jsou rovněž koncové, což jasně vyjadřuje, že konec těchto dat je ovlivněn pouze a jen okamžikem, ve kterém byla stažena. Mohlo by nás napadnout, že podobná návaznost a tedy i smyčka by měla existovat i u objektů „*AgentTerminated*“ a „*AgentStarted*“. Zdá se logické, že pokud jsme z agenta soubor získali, tak musel být nastartován. A tak by soubor teoreticky neměl končit objektem „*AgentTerminated*“. Musíme si však uvědomit, že komunikační vlákno agenta je implementováno tak, že je schopno přenášet soubor i nějakou chvíli po ukončení agenta. Proto se k nám mohou dostat data, jejichž posledním objektem bude skutečně „*AgentTerminated*“.

## 6.3 Klient pro prezentaci výsledků

Poslední část, kterou by měl systém obsahovat, je klient pro prezentaci výsledků. Jak jsme již nastínili v předchozích kapitolách, tuto část lze realizovat dvěma způsoby. Můžeme vytvořit aplikaci s uživatelským rozhraním, která bude mít schopnost připojit se do databáze a pomocí nejrůznějších operací prezentovat výsledky. Šlo by o aplikaci postavenou na ADO.NET, což je soubor tříd

umožňující přístup k datům. Druhým řešením je vytvoření webové aplikace (intranetového systému) ASP.NET. Samozřejmě, že i toto řešení by bylo postaveno na principu připojení do databáze a získávání a prezentaci dat pomocí dotazů SQL. Opět bychom proto využívali ADO.NET a z tohoto pohledu by se tato dvě řešení příliš nelišila.

Zejména z důvodu implementace byla zvolena druhá možnost a byla vytvořena webová aplikace v principu podobná obyčejným webovým stránkám. Zkrátka a dobře ASP.NET se více hodí k prezentaci dat formou, jakou potřebujeme v našem projektu. Mnoho prvků a operací, které bychom jinak museli implementovat „ručně“, je v ASP.NET již pevně zakomponováno. Příkladem může být třídění sloupců, inteligentní ovládací prvky, pomocí kterých lze rychle a efektivně zpracovat získaná data apod. Na druhou stranu, abychom stále jen nevychvalovali, vše má i své nevýhody. Pokud například potřebujeme vytvořit něco, co není úplně obvyklé, a co není přímo součástí nástrojů ASP.NET, tak je implementace o něco složitější. Další věc, kterou můžeme označit za nevýhodu je, že pro ASP.NET aplikace samozřejmě potřebujeme webový server IIS. Dobře řešenou částí u webových aplikací je integrované přihlašování do aplikace. Spouštěná webová aplikace sama zjistí, kdo ji spustil a podle toho i práva této osoby, protože login v databázi se vždy musí shodovat s loginem účtu ve Windows. Spustit tuto aplikaci je navíc dovoleno pouze osobám, které jsou administrátory jak v databázi, tak při přihlášení do Windows.




***Obr. 14*** *– Princip architektury model-view-controller*

Ještě než se pustíme do popisu samotného klienta, řekneme si něco o architektuře MVC („*model-view-controller*“), kterou ASP.NET aplikace splňují. Tato architektura je postavena na principu oddělení datové logiky („*model*“), uživatelského rozhraní („*view*“) a logiky zpracování požadavků („*controller*“). Z pohledu této architektury funguje naše webová aplikace tak, že uživatel

provede nějakou akci na uživatelském rozhraní a předá řízení logice pro zpracování požadavků. „*Controller*“ přistoupí ke komponentě zvané „*model*“, což je pouze synonymum pro doménovou vrstvu, která přímo odpovědná za zpracování dat. V posledním kroku pak komponenta „*view*“ zobrazí aktualizovaná data, a to již nezávisle na komponentě „controller“. Celý princip je zobrazen na obr. 14.

V následujících podkapitolách se budeme věnovat nejprve prezentaci dat z pohledu uživatele, tj. co vlastně aplikace zpracovává, a posléze se podíváme na implementaci aplikace, tj. jak to zpracovává.

### 6.3.1 Prezentace výsledků, vyhodnocování dotazů

V kapitolách, ve kterých jsme diskutovali o požadavcích na vyvíjený systém, jsme si řekli, že výsledky, které musí systém poskytovat, musí být především informace o využití software a informace o činnosti pracovníků. Oba tyto základní požadavky jsou splněny.

Využití softwaru může být kontrolováno ze dvou hledisek. Aplikace jednak obsahuje aktivitu osob, ze které je možné vyčíst, co který pracovník spouštěl, jak dlouho program běžel apod., ale především aplikace obsahuje formulář s názvem „*Využití sw*“. Tento formulář má dvojí funkci. Můžeme vyplnit název aplikace, kterou chceme kontrolovat, datum od kdy do kdy chceme kontrolu provést, a případně vybrat pracovníka, pro kterého chceme tyto informace zjistit. Výsledkem je tabulka, která zobrazí název této hledané aplikace, název procesu, ke kterému je vázána, výrobce a samotné výsledky v podobě časových údajů. Prvním údajem je celková doba běhu této aplikace, tj. jak dlouho proces spojený s aplikací skutečně existoval. Druhým údajem, který se dozvíme, je celková doba běhu této aplikace na popředí. A konečně posledním údajem je celková doba, po kterou byl uživatel, popř. všichni uživatelé, neaktivní v této aplikaci. Z těchto tří údajů může administrátor jasně určit, jak je příslušný software využíván a popřípadě i vyvodit důsledky. Můžeme ale požadovat, aby systém zpracoval všechny aplikace nainstalované na všech stanicích v jedné tabulce. V takovém případě nechá administrátor kolonku s názvem aplikace prázdnou a zbytek vyplní, jak potřebuje. Klient zpracuje dotaz tak, že zobrazí informace pro všechny instalované produkty.

Další, co musí klient prezentovat, je činnost uživatelů. K tomu slouží část aplikace zvaná „Aktivity osob“, která zobrazuje seznam pracovníků. Vybereme-li si některého pracovníka, tak si zvolíme den, na který se chceme podívat a nakonec vybereme, zda chceme vidět procesy nebo aplikace. V drtivé většině případů se budou kontrolovat aplikace. Zobrazení aplikací je v tabulce, kde nejdůležitějšími informacemi jsou nadpis okna, od kdy do kdy bylo okno aktivní, a doba po kterou byl v okně uživatel neaktivní. Takto „rozházená“ data jsou poměrně nepřehledná, ovšem velkou výhodou je, že můžeme snadno tyto informace řadit podle sloupců. Okamžitě tak vidíme např. kdy

byl uživatel nejvíce neaktivní apod. U každého řádku je navíc možnost zobrazit více informací, které ukazují další informace o procesu, ke kterému se aplikace váže.

To by však mohlo být poměrně málo, a tak je aplikace obohacena částí „Vyhledávání a filtry“. Pomocí formuláře, který obsahuje, lze vyhledávat záznamy podle nadpisu okna nebo názvu aplikace, a přitom můžeme zadat i hodnotu neaktivity, která vyhledá pouze záznamy, které mají vyšší než zadanou neaktivitu. Navíc velkou výhodou je, že nemusíme zadat přesný název, který byl v nadpisu příslušného okna, ale pokud např. zadáme do názvu pouze slovo „hry“, vyhledá nám všechna okna, která toto slovo obsahovala ve svém nadpisu. Tím můžeme velmi rychle a efektivně odhalovat pracovníky, kteří se věnují jiným než pracovním činnostem. Možnost omezit výběr podle času a podle počítače je v tomto formuláři samozřejmostí.

Z předchozích řádků možná není úplně jasné, že funkci řazení jednotlivých sloupců lze využít téměř všude. Aplikace v ASP.NET umožňuje použít tento mechanismus u každé tabulky, a tak, i když se jedná o tabulku, která byla vytvořena na základě nějakého dotazu pomocí formuláře, můžeme ji rovněž řadit podle sloupců. Součástí této aplikace je i správa databáze. Stejnojmenná část aplikace je pak schopna mazat záznamy v zadaném časovém rozmezí se zohledněním toho, jak chceme mazat. Můžeme promazat buď údaje pro konkrétní počítač nebo pro všechny počítače, anebo stejně tak pro konkrétního pracovníka nebo pro všechny pracovníky. Je ale velmi důležité si uvědomit, jaké může mít takové promazání následky.

Agent funguje tak, že vždy zaznamená běžící procesy a posléze zaznamenává okna, u kterých kromě všech běžných informací o okně samotném ukládá identifikátor procesu, který okno vytvořil. Potom tedy kolektor pouze hledá v databázi proces podle tohoto identifikátoru. Pokud ovšem stáhneme data z agenta, tak se zpravidla jedná o chvíli, kdy agent běží a pravděpodobně budou po tomto stažení následovat další aktivní okna. To však znamená, že jestliže po tomto prvním stažení dat promažeme databázi, tak další stažená data budou vypadat tak, že okna nebudou mít své procesy uloženy v databázi, a tím pádem bude tato vazba porušena. Je proto třeba mazat s rozmyslem a to pouze data taková, u kterých nehrozí, že by došlo k podobným anomáliím.

Součástí správy databáze je pochopitelně i správa osob a správa počítačů. Je velmi intuitivní, a tak si pouze povíme o věcech, které by nás mohly překvapit. Co se týká osob, tak je dobré vědět, že nelze smazat osobu, pokud je v systému posledním administrátorem. Je to celkem logické, protože použít aplikaci pro prezentaci výsledků může pouze administrátor, a tak pokud by byl poslední administrátor odstraněn, nebylo by možné systém spustit. Dále by nás mohlo překvapit chování systému v okamžiku mazání počítačů nebo osob. V obou případech se smazaná data z tabulek neztratí, ale pouze se vypíše datum smazání osoby nebo počítače. Po smazání dojde pouze k tomu, že

ze smazaných stanic se posléze nestahují data a pracovníci se nemohou přihlásit do aplikace pro zobrazení výsledků. Smazané položky lze snadno obnovit odstraněním doby, kdy byla položka smazána.

### 6.3.2 Implementace webové aplikace pro zobrazení výsledků

V této kapitole si popíšeme důležité prvky aplikace z pohledu implementace. Zaměříme se na některé významné dotazy SQL, které budou k vidění v příloze 1. Tento popis provede z pohledu jednotlivých souborů a to pouze stručným přehledem, přičemž podrobný popis je k vidění v příloze 2. V následující tabulce je popis souborů, které jsou slouží zpravidla ke konfiguraci, nebo obsahují globální proměnné apod.

| Název souboru | Stručný popis |
|---|---|
| Web.Config | Soubor obsahuje konfigurační nastavení |
| Global.asax | Soubor obsahuje metody zajišťující reakce na základní události |
| MonitoringPresentationLayerLib.cs | Soubor obsahuje metody sloužící ke kontrole různých hodnot |
| CurrentUser.cs | Soubor obsahuje metody pro nastavení session* |
| MasterPage.master | Soubor slouží jako hlavní stránka, do které se pouze generuje obsah |

#### Webové ovládací prvky – Web User Control

Velká síla webových aplikací napsaných v ASP.NET spočívá v možnosti vytvoření vlastních webových ovládacích prvků. Pokud máme prvek, o kterém víme, že se bude často opakovat, a že jeho funkci jsme schopni využít vícekrát, tj. na více stránkách, vytvoříme si ovládací prvek, který pak z ostatních stránek pouze načteme. Samozřejmě málo které dvě stránky využijí ovládací prvek naprosto stejně, a proto musí být do určité míry univerzální. Při tvorbě tohoto prvku je na nás, abychom rozhodli, na kolik jsou určité stránky, kde chceme prvek využít, odlišné, a zda se nám prvek vyplatí vytvářet z hlediska pracnosti anebo je snazší udělat si každou stránku zvlášť. Soubory, které jsou těmito ovládacími prvky tentokrát popíšeme přímo v této kapitole. U popisu dalších souborů v příloze 2 ovšem budeme na tyto soubory často odkazovat.

* Český překlad by mohl být „sezení“, ale termín je natolik užívaný, že se ho budeme nadále držet.

Na příkladu našeho projektu bude jasně vidět výhoda ovládacího prvku. Naše webová aplikace obsahuje dva ovládací prvky. Prvním z nich je „*DetailsOfMonitoring.ascx*“, který zobrazuje detaily pro vysledovaná okna (soubor „*DetailsOfWindow.aspx*“), pro procesy (soubor „*DetailsOfProces.aspx*“) a pro nalezená okna ve výsledcích vyhledávání (soubor „*DetailsOfResultOfSearching.aspx*“). Druhým ovládacím prvkem je „*ShowPersons.ascx*“, který zobrazuje osoby v oblasti aktivit pracovníků (soubor „*PersonsActivities.aspx*“) a v oblasti správy databáze (soubor „*AdminPerson.aspx*“).

Prvek pro zobrazování detailů („*DetailsOfMonitoring.ascx*“) je navržen tak, aby zobrazil podrobné informace o okně, o procesu a o počítači, ke kterému jsou tyto informace vázány. V případě, že však zobrazuje podrobnosti pouze o procesu, tak skryje tabulku s podrobnostmi o okně. V ovládacím prvku totiž můžeme vytvořit něco, co nám rozhoduje o tom, v jakém módu se zrovna nacházíme. Na základě tohoto módu je vytvořen i dotaz SQL a rozložení tabulek a textů v tomto ovládacím prvku. Stránka, která prvek využívá jej potom volá v některém z přednastavených módů.

Prvek pro zobrazování pracovníků „*ShowPersons.ascx*“ je postaven velmi podobně. Rovněž obsahuje svůj mód, protože při zobrazení seznamu pracovníků v oblasti aktivit (stránka „*PersonsActivities.aspx*“) vidíme pouze aktivní pracovníky, zatímco v části pro správu databáze (soubor „*AdminPersons.aspx*“) lze vidět pracovníky všechny. Tím je myšleno, že jsou zobrazováni i smazaní pracovníci a tabulka navíc obsahuje záznam data, kdy byli ze systému odstraněni. Tento ovládací prvek, kromě tabulky zobrazující pracovníky, navíc obsahuje seznam, ze kterého můžeme snadno zvolit konkrétního pracovníka (tj. zobrazit řádek tabulky s informacemi o tomto pracovníkovi).

Tímto bychom měli vyřešeny ovládací prvky. Konkrétními dotazy SQL jsme se zatím nezabývali, protože u obou ovládacích prvků se zpravidla jedná o triviální dotazy.

### Webové formuláře – Web Forms

V ASP.NET se všechny stránky nazývají webové formuláře bez ohledu na to, zda skutečně nějaký formulář obsahují. V této kapitole se zaměříme na stručný popis činnosti souborů, o kterých jsme prozatím nemluvili. Soubory, probírané v předchozí podkapitole o ovládacích prvcích, budou zmíněny pouze okrajově. Podrobný popis těchto souborů opět najdeme v příloze 2. Popis prvního probíraného souboru („*ListOfDays.aspx*“) bude v příloze 2 obsáhlejší v tom smyslu, že se více zaměříme na věci, které jsou používány i v ostatních souborech. Další soubory proto nebudou rozebírány do takových podrobností. Následuje stručný popis souborů (které nevyužívají webové ovládací prvky probrané dříve).

| Název souboru | Stručný popis |
|---|---|
| ListOfDays.aspx | Soubor slouží pro zobrazení dní, ve kterých byly uloženy záznamy pracovníků |
| ActivitiesOfDayWindow.aspx | Soubor slouží pro zobrazení aktivních oken |
| ActivitiesOfDayProcess.aspx | Soubor slouží pro zobrazení procesů |
| Searching.aspx | Soubor obsahuje formulář sloužící k vyhledávání aktivních oken |
| ResultOfSearching.aspx | Soubor zobrazuje výsledky vyhledávání |
| SwUsing.aspx | Soubor obsahuje formulář sloužící k vyhodnocení používání softwaru na stanicích |
| ResultOfUsingSw.aspx | Soubor zobrazuje výsledky použití softwaru |
| AdminDatabase.aspx | Soubor pouze zobrazuje lokální menu u správy databáze |
| AdminDeleteItems.aspx | Soubor obsahuje formulář, který slouží k nastavení odstraňovaných položek |
| ResultDeleting.aspx | Soubor slouží pouze ke zobrazení výsledků mazání |
| AdminPersonDetails.aspx | Soubor slouží k zobrazení detailů osob |
| AdminComputers.aspx | Soubor slouží pro zobrazení seznamu stanic |
| AdminComputerDetails.aspx | Soubor zobrazuje detaily stanic |

## 6.4 Ostatní součásti a nastavení systému

Součástí systému jsou i menší aplikace, které jsou rovněž důležité, a proto se jim budeme věnovat v této kapitole. Obsahem této kapitoly bude tvorba databáze a z toho plynoucí tvorba instalátoru pro databázi a tvorba instalátorů agenta a kolektoru a jejich nastavení.

Visual studio 2005, ve kterém byly implementovány všechny části systému, obsahuje prostředky pro tvorbu instalátorů. Neobsahuje ovšem žádný prostředek pro tvorbu instalátoru databáze. Instalátor databáze není nic jiného než aplikace, jejímž cílem je vytvoření databáze. Protože však instalátor musíme vytvořit sami, používáme pro procházení jednotlivých kroků instalátoru překrývání jednotlivých vrstev uživatelského rozhraní. Každá vrstva, neboli panel, má vlastnost, díky které nastavíme viditelnost této vrstvy. Řízení těchto panelů bychom mohli opět popsat konečným

automatem, který vždy zjistí, že bylo stisknuto tlačítko „*další*“ nebo „*zpět*“. Zjistí si, kde se nachází, a na základě toho přejde do jiného stavu. Jak procházíme jednotlivá nastavení tohoto instalátoru, skrýváme nebo zobrazujeme jednotlivé panely aplikace. V instalátoru je nejprve nutné nastavit server SQL (tj. název počítače, na kterém SQL server běží) a instanci serveru (název serveru). V dalším kroku je třeba rozhodnout, jak se bude databáze jmenovat a zvolit také typ autentizace. Jednak lze využít integrovanou autentizaci, která je stejná jako přihlášení do Windows. V takovém případě nemusíme nic zadávat. Druhou možností je autentizace serverem SQL, kde musíme na serveru SQL vytvořit účet, a potom login a heslo v tomto kroku vyplnit i v instalátoru. Následuje žádost o vyplnění jména a loginu administrátora systému, kde login musí být shodný s loginem administrátora stanice, kde bude umístěn kolektor. Další obrazovka nám umožní zkontrolovat zadané údaje a pokračuje v připojení k serveru SQL.

Vše je implementováno tak, že se vytvoří dvě vlákna, kde první pracuje s tzv. status barem a druhé vlákno se připojuje. Vlákno používané pro připojení je založeno na připojovacím řetězci (tzv. „*connection string*“), který je složen z údajů, které uživatel v instalátoru vyplnil. Pokud dojde k úspěšnému připojení, tak instalátor vytvoří databázi pomocí přednastavených dotazů SQL. Instalátor pak oznámí úspěšnou instalaci a je ukončen.

Další součásti systému jsou instalátory agenta a kolektoru, jejichž tvorba je mnohem snazší. Visual Studio 2005 totiž obsahuje možnost vytvoření projektu, který má název „*Setup Project*“, což je samotný instalátor, který pouze vhodně nastavíme. Nad instalátorem se pouze zvolí, který projekt se má instalovat. Takto jsou vytvořeny oba instalátory – pro agenta i kolektor.

Více se ale zmíníme o nastavení agenta i kolektoru. Pro nastavení agenta slouží soubor „*Agent.exe.config*“ a pro nastavení kolektoru soubor „*Server.exe.config*“. Tyto soubory jsou napsány v jazyce XML. Agent umožňuje nastavit dvě proměnné. První proměnná nese název „*TimerInterval*“ a určuje v milisekundách periodu, s jakou se prochází monitorovací vlákno. V následujícím příkladě vidíme nastavení této periody na 2 sekundy.

```
<setting name="TimerInterval" serializeAs="String">
      <value>2000</value>
</setting>
```

Druhá proměnná má název „*TimeInactive*“ a jde o nastavení doby, po které začne agent sledovat, zda je uživatel v okně neaktivní. Příklad nastavení této proměnné na 3 sekundy je zde:

```
<setting name="TimeInactive" serializeAs="String">
     <value>3</value>
```

```
</setting>
```

Důvodem proč je hodnota jednou v sekundách a podruhé v milisekundách je, že záleží na tom, do jaké proměnné se v kódu agenta předává.

Proměnné, které lze nastavit u serveru je „*connectionString*“, která slouží k nastavení spojení s databází:

```
<connectionStrings>
      <add name="Alc.Monitoring.Server.Properties.Settings.Db"
      connectionString="DataSource=FIT-DELL-
      LAPTOP\SQLEXPRESS;Initial Catalog=Monitoring;Integrated
      Security=True" />
</connectionStrings>
```

Druhou proměnnou je „*TransferDelayMinutes*“, jejíž hodnota je v minutách a určuje periodu, se kterou se kolektor pokouší stáhnout data ze stanic:

```
<setting name="TransferDelayMinutes" serializeAs="String">
      <value>1440</value>
 </setting>
```

A konečně třetí a poslední proměnná „*TransferTryDelayMinutes*“ je opět v minutách a určuje dobu, po které se server opět pokusí stáhnout data ze stanic v případě, že předchozí pokus dopadl neúspěšně:

```
<setting name="TransferTryDelayMinutes" serializeAs="String">
      <value>15</value>
</setting>
```

V neposlední řadě si musíme říct o nastavení firewallu a verzích platformy .NET a jejím využití. Může dojít k tomu, že firewall zamezí komunikaci agent – kolektor. Proto musí být přidána výjimka na firewallu stanice, na které je agent nasazen. Při testování systému bylo rovněž zjištěno, že kompatibilita verzí platformy .NET bohužel není nejlepší. Když je na stanici nainstalována verze .NET 2.0, agent monitoruje naprosto bez problému. Pokud je však instalována verze vyšší, agent není schopen monitorovat aktivní okna. Je proto nutné dodržet předepsanou verzi této platformy. Nekompatibilita plyne z toho, že k tomu, aby mohla služba monitorovat aktivní okna, musí mít aktivovaný tzv. přístup k ploše. Pokud se podíváme na vlastnosti služeb operačního systému

Windows XP, tak na záložce „Přihlášení“ je možnost zatrhnout „Povolit službě používání plochy“. My to provádět nemusíme, protože agent si to po instalaci aktivuje sám. Problémem ale je, že v operačním systému Windows Vista, který je právě na platformě .NET verze 3.5 založen, již možnost aktivovat přístup k ploše u služeb nemáme. Proto pravděpodobně dochází ke zmiňovaným potížím.

Řešením v systému Windows Vista by byla aplikace, která by používala skryté okno. Byla by tedy tvořena jako aplikace s uživatelským rozhraním a její okno by bylo skryto, ale mělo by přístup k ostatním oknům. Tímto přístupem bychom byli pravděpodobně schopni dosáhnout stejných výsledků jako se stávající službou agenta. Částečně již byla tato metoda testována, ovšem její plná realizace by mohla být otázkou možných rozšíření systému.

## 6.5 Možná implementační rozšíření a slabiny systému

Další možnosti, jak stávající systém zlepšit, budeme řešit i v této kapitole. Jednou z možných zlepšení může být využití skenovaných dat, díky kterým lze snadno vytvořit docházkový systém. Z předchozích kapitol víme, že z objektu „*LoginPerson*“ pouze získáme identifikátor uživatele a nezajímá nás čas jeho přihlášení. Pokud bychom zpracovávali i čas přihlášení, bylo by snadné přidat jednu tabulku do databáze, a pomocí tohoto systému tak zaznamenávat docházku pracovníků. Náš systém je však součástí produktu, který již obsahuje software, který se na docházku přímo specializuje, a tak by bylo toto rozšíření nadbytečné. Prakticky lze kontrolu docházky pracovníka provádět i v současné implementaci. Stačí se pouze podívat na první a poslední aktivní okna, která zpravidla určují čas přihlášení a odhlášení pracovníka. Nejedná se však o skutečný docházkový systém.

Rozšíření, které by však mělo skutečný význam, je implementace šifrování probraná v kap. 5.3.1, protože by došlo k značnému zlepšení bezpečnosti citlivých dat. Za zmínku by stála i možná změna přenosového protokolu TCP na UDP, což by zase přineslo zlepšení z pohledu přenosu dat. Tato změna by ovšem vyžadovala zásadní změnu kolektoru.

Slabinou systému je, že pracovník si může otevřít okno, které zmenší. Tím pádem může sledovat obsah okna, které je pod ním, a přitom bude za aktivní okno považované to zmenšené. Tímto přístupem může dojít ke zkreslení vysledovaných informací. Je zřejmé, že uživatel by podobnou činnost prováděl v případě, kdy by si nepřál, abychom zjistili, že takovou aplikaci používal. Dobu běhu této aplikace však můžeme zjistit alespoň pomocí procesu, ke kterému se váže. Definitivní řešení tohoto problému by mohlo být založeno na dodatečném sledování velikosti a pozice okna [7].

# 7 Případová studie použitelnosti

V této kapitole se zaměříme na praktické využití našeho systému a pokusíme se ověřit, zda jsme splnili zadání projektu se všemi jeho detaily. Případová studie však netestuje systém tak, jako se tomu děje v ostrém provozu.

Bohužel počet stanic, na kterých můžeme systém nasadit je poměrně nízký (jedná se o tři stanice). Proto budeme diskutovat i o tématech jako jsou rozšiřitelnost, škálovatelnost apod. Ukážeme si konkrétní výsledky a konkrétní data, která nám systém poskytne. Budeme tak moci zhodnotit, do jaké míry splňuje požadavky, které jsme stanovili v počátcích vývoje projektu a zaměřit se i na faktory, které bychom mohli považovat za nedostatky. V neposlední řadě musíme na konkrétních výsledcích ukázat, co nám vlastně říkají. Zjistit, jestli jde o dostatečně relevantní informace, na základě kterých bychom mohli ve firmě podniknout kroky, jenž by mohly šetřič čas i peníze.

## 7.1 Parametry případové studie

Jak již bylo řečeno, budeme sledovat chování našeho systému v síti se třemi stanicemi a třemi uživateli („pracovníky“). Na každou stanici bude nainstalován agent a jedna z těchto stanic nám poslouží současně jako server, kde poběží kolektor a databáze SQL. Tato stanice bude sloužit i pro prezentaci vysledovaných informací a správu databáze. Monitorovat budeme tyto stanice tři dny, a potom bude následovat vyhodnocení vysledovaných informací.

Nastavení agentů bude takové, že monitorovacím vláknem se prochází každé dvě vteřiny a neaktivita uživatele v příslušném okně se začne započítávat po třech vteřinách. Kolektor bude stahovat data ze stanic každých 1440 minut, což je 24 hodin. Tyto hodnoty jsme zvolili proto, že podobné nastavení předpokládáme i při nasazení v praxi. V případě neúspěšného pokusu o stažení dat se bude pokus opakovat po patnácti minutách. Vytížení stanic můžeme popsat jen hrubým odhadem. První stanice je aktivně používána přibližně dvanáct hodin denně. Aktivní používání druhé stanice je velmi podobné a třetí stanice je aktivně používána přibližně dvě hodiny denně. Ještě zbývá dodat, že všechny tři stanice používají operační systém Windows XP Professional.

Abychom na tomto příkladu ukázali, že systém splňuje zadané požadavky, zaměříme se ve výsledcích především na sledování konkrétních uživatelských aktivit a vyhledávání v těchto

aktivitách. Druhou a neméně důležitou činností, kterou budeme sledovat, bude využití softwaru na těchto stanicích.

## 7.2 Získané výsledky

Po třech dnech sledování přichází na řadu zhodnocení vysledovaných informací. Nejprve se podíváme zcela samostatně na výsledky sledování bez vyhledání některých podrobností. Ještě než k tomu dojdeme, chceme upozornit, že byly záměrně odstraněny sloupce tabulky (odkaz na podrobnosti, AdGuid počítače, ID počítače), z důvodu kvalitnějšího zobrazení náhledu. Tyto sloupce nejsou pro případovou studii důležité.

**Jméno:** Aleš Skopal **Kancelář:** L01 - Programator **Datum:** 2009.02.25

| Nadpis | Aktivní od | Aktivní do | Doba neaktivity osoby (min.) | Název aplikace | Výrobce aplikace | Jméno počítače |
|---|---|---|---|---|---|---|
| Compose: Connect! - asset ma | 25.2.2009 14:17:43 | 25.2.2009 14:17:45 | 0 | Thunderbird | Mozilla Corporation | FIT-DELL-LAPTOP |
| Compose: Connect! - asset managament | 25.2.2009 14:17:45 | 25.2.2009 14:17:47 | 0 | Thunderbird | Mozilla Corporation | FIT-DELL-LAPTOP |
| Sent - Thunderbird | 25.2.2009 14:17:47 | 25.2.2009 14:17:49 | 0 | Thunderbird | Mozilla Corporation | FIT-DELL-LAPTOP |
| AuditPro centrum Praha - Mozilla Firefox | 25.2.2009 14:17:49 | 25.2.2009 14:17:53 | 0 | Firefox | Mozilla Corporation | FIT-DELL-LAPTOP |
| Compose: Connect! - asset managament | 25.2.2009 14:17:53 | 25.2.2009 14:20:25 | 0,65 | Thunderbird | Mozilla Corporation | FIT-DELL-LAPTOP |
| Message Compose | 25.2.2009 14:20:25 | 25.2.2009 14:20:27 | 0 | Thunderbird | Mozilla Corporation | FIT-DELL-LAPTOP |
| Compose: Connect! - asset managament | 25.2.2009 14:20:27 | 25.2.2009 14:20:29 | 0 | Thunderbird | Mozilla Corporation | FIT-DELL-LAPTOP |
| Mail Server Password Required | 25.2.2009 14:20:29 | 25.2.2009 14:20:33 | 0 | Thunderbird | Mozilla Corporation | FIT-DELL-LAPTOP |
| Sent - Thunderbird | 25.2.2009 14:20:33 | 25.2.2009 14:20:37 | 0 | Thunderbird | Mozilla Corporation | FIT-DELL-LAPTOP |
| Inbox - Thunderbird | 25.2.2009 14:20:37 | 25.2.2009 14:20:41 | 0 | Thunderbird | Mozilla Corporation | FIT-DELL-LAPTOP |
| AuditPro centrum Praha - Mozilla Firefox | 25.2.2009 14:20:41 | 25.2.2009 14:20:45 | 0 | Firefox | Mozilla Corporation | FIT-DELL-LAPTOP |
| AssetManagament.odt - OpenOffice.org Writer | 25.2.2009 14:20:45 | 25.2.2009 14:20:47 | 0 | | OpenOffice.org | FIT-DELL-LAPTOP |

***Obr. 15*** – *Část výpisu aktivit uživatele*

Na obr. 15 tedy můžeme vidět, že se pracovník v daném okamžiku věnoval tvorbě článku do časopisu s tématem „*asset management*“. Toto téma je shodou okolností velmi blízké našemu tématu, tj. personálnímu auditu. Je dobré zmínit, že počet záznamů u nejvytíženější stanice dosahuje okolo 2000 oken za den. Proto je nutné, využívat vyhledávacích formulářů. A tento formulář se pokusíme využít právě teď, kdy budeme zjišťovat, zda se někteří uživatelé nevěnovali hraní her. Vyplnění formuláře pro vyhledávání je naprosto snadné, pouze napíšeme do názvu aplikace: „*hry*“. Opět se

sluší říct, že jsme odstranily sloupce (ID pracovníka, ID stanice, odkaz na podrobnosti), které nebyly nezbytné. Z obr. 16 jsme nakonec zjistili, že v naší síti byli uživatelé, kteří se věnovali hraní her.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Jiří Skopal | Onlinehry.cz - hrajte online hry zdarma - Mozilla Firefox | 26.2.2009 13:34:11 | 26.2.2009 13:34:37 | 0,317 | Firefox | Mozilla Corporation | Zfp-machine |
| Jiří Skopal | Onlinehry.cz - hrajte online hry zdarma - Mozilla Firefox | 26.2.2009 13:34:41 | 26.2.2009 13:34:53 | 0,133 | Firefox | Mozilla Corporation | Zfp-machine |
| Jiří Skopal | Počítačové hry - Seznam - Mozilla Firefox | 26.2.2009 13:34:53 | 26.2.2009 13:34:55 | 0 | Firefox | Mozilla Corporation | Zfp-machine |
| Aleš Skopal | magazín eKamarád \| Jak vytvořit češtinu do hry nebo programu - Mozilla Firefox | 25.2.2009 23:13:53 | 25.2.2009 23:13:58 | 0 | Firefox | Mozilla Corporation | FIT-DELL-LAPTOP |
| Aleš Skopal | Online hry - Herní Ostrov - Mozilla Firefox | 25.2.2009 23:17:39 | 25.2.2009 23:17:49 | 0,083 | Firefox | Mozilla Corporation | FIT-DELL-LAPTOP |
| Aleš Skopal | Online hry - Herní Ostrov - Mozilla Firefox | 25.2.2009 23:17:53 | 25.2.2009 23:17:59 | 0 | Firefox | Mozilla Corporation | FIT-DELL-LAPTOP |

***Obr. 16** – Část výpisu po vyhledání hráčů her*

Nyní se pokusíme najít aplikaci, ve kterých byli pracovníci neaktivní více než 30 minut. Jde opět o velmi snadnou záležitost, kdy ve vyhledávacím formuláři zadáme pouze hodnotu uživatelské neaktivity. Obr. 17 zobrazuje výsledek, ve kterém vidíme, že jeden z uživatelů dokonce lenivěl i při tvorbě diplomové práce. Dále by nás mohla napadnout otázka, proč jsou některá pole prázdná. V třetím řádku je to proto, že některé procesy nám neprozradí výrobce aplikace. Ovšem v řádku druhém, kde jsou prázdné hned tři buňky, měl uživatel v danou chvíli aktivní pouze plochu, nebo spořič obrazovky. Zbývá jen zmínit, že jsme vypustili stejné sloupečky, jako v předchozím příkladě. Názvy sloupců byly rovněž odstraněny z důvodu lepšího zobrazení tabulek.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Jiří Skopal | InterVideo WinDVD 5 | 24.2.2009 22:51:45 | 24.2.2009 23:23:03 | 31,067 | WinDVD Application | InterVideo Inc. | Zfp-machine |
| Jiří Skopal | | 25.2.2009 12:23:16 | 25.2.2009 13:59:20 | 95,967 | | | Zfp-machine |
| Aleš Skopal | diplomka.odt - OpenOffice.org Writer | 24.2.2009 20:08:44 | 24.2.2009 20:47:34 | 38 | | OpenOffice.org | FIT-DELL-LAPTOP |
| Aleš Skopal | Norton Security Scan | 25.2.2009 18:05:57 | 25.2.2009 20:19:05 | 133,05 | Standalone Scanner Components | Symantec Corporation | FIT-DELL-LAPTOP |

***Obr. 17** – Výpis při vyhledání oken, ve kterých byla neaktivita větší než třicet minut*

Zaměřme se nyní na zjišťování využití softwaru. Jistě bychom mohli procházet jednotlivé výsledky a na jejich základě rozhodovat o využitelnosti některých programů. To by bylo ovšem značně komplikované. Proto využijeme formulář, který je schopen jednotlivé výsledky agregovat. Řekněme, že budeme chtít sledovat nejprve využití internetového prohlížeče Firefox. Na obr. 18

vidíme výsledky využití Firefoxu konkrétním pracovníkem. Zadání formuláře je opět velmi jednoduché. Vybereme pracovníka a napíšeme název aplikace, jejíž využití chceme zjistit.

**Využití sw** - Výpis aplikací pro osobu: Aleš Skopal

| Název aplikace | Orig. název souboru | Výrobce programu | Celková doba běhu aplikace (min.) | Celková doba běhu aplikace na popředí (min.) | Celková doba neaktivit v aplikaci (min.) |
|---|---|---|---|---|---|
| Firefox | firefox.exe | Mozilla Corporation | 1355 | 462 | 211,124 |

***Obr. 18** – Výpis využití Firefoxu konkrétní osobou*

Pokud se chceme dozvědět o využití tohoto prohlížeče na všech počítačích, jednoduše zvolíme všechny uživatele. Výsledek můžeme vidět na obr. 19.

**Využití sw**

| Název aplikace | Orig. název souboru | Výrobce programu | Celková doba běhu aplikace (min.) | Celková doba běhu aplikace na popředí (min.) | Celková doba neaktivit v aplikaci (min.) |
|---|---|---|---|---|---|
| Firefox | firefox.exe | Mozilla Corporation | 1753 | 790 | 428,130000000001 |

***Obr. 19** – Výpis využití Firefoxu všemi uživateli*

A konečně se podívejme na poslední zkrácený výpis, který je výsledkem zpracování všech počítačů a všech aplikací na nich nainstalovaných. Tento výsledek nám systém poskytne, pokud ve formuláři pro zjištění využití software pouze stiskneme tlačítko bez vyplnění formuláře. Výsledek tohoto vyhledání je k vidění na obr. 20. Názvy sloupců jsou opět vynechány, ale jde o stejné tabulky jako na obr. 18 a obr. 19.

| | | | | | |
|---|---|---|---|---|---|
| Microsoft(R) Windows (R) 2000 Operating System | taskmgr.exe | Microsoft Corporation | 184 | 2 | 0,184 |
| Microsoft(R) Windows (R) Operační systém | UPDATE.EXE | Microsoft Corporation | 2 | | |
| Microsoft(R) Windows Media Player | setup_wm | Microsoft Corporation | 13 | 13 | 13,667 |
| Microsoft(R) Windows Media Player | WMPLAYER.EXE | Microsoft Corporation | 66 | 22 | 20,85 |
| Microsoft® Visual Studio® 2005 | csc.exe | Microsoft Corporation | 13 | | |
| Microsoft® Visual Studio® 2005 | devenv.exe | Microsoft Corporation | 70 | 2 | 0,75 |

***Obr. 20** – Část výpisu využití všeho softwaru na všech počítačích*

Měli bychom si vysvětlit i důvod, proč se v tabulce vyskytují prázdná políčka. Je tomu tak proto, že v tomto případě sledujeme produkty nezávisle na tom, zda mají uživatelská rozhraní. Proto se ve výpisu objevují i procesy, které se pochopitelně nemohou na popředí dostat.

## 7.3 Zhodnocení studie

Díky provedené případové studii, jsme ověřili, že systém naprosto splňuje zadání a požadavky, a je možné jej nasadit v praxi. Prověřili jsme funkčnost celého systému od prvotní instalace a monitorování stanic, až po zhodnocení vysledovaných výsledků, pomocí formulářů. Zjistili jsme, že je velmi snadné odhalovat pracovníky, kteří se věnují jiným než pracovním činnostem, a rovněž jsme ověřili, jak snadno můžeme zjišťovat, využití softwarových produktů.

Bohužel počet stanic, na kterých jsme mohli tuto studii provést je velmi malý. Můžeme se však alespoň zamyslet nad množstvím získaných dat. Je zřejmé, že velikost dat je přímo úměrná vytíženosti stanice. Pokud pracovník často přepíná okna nebo při surfovaní po internetu mění nadpis okna, tak množství dat bude větší než u pracovníků, kteří jsou méně aktivní. Průměrná velikost souboru za jednu hodinu je asi 150 kB. Při osmi-hodinové pracovní době jde přibližně o 1200 kB. Budeme-li uvažovat síť se sto počítači, musíme každý den stáhnout přibližně 120 MB dat. Můžeme konstatovat, že se nejedná o nijak velké množství dat v případě, že administrátor bude pravidelně databázi promazávat. Studie tedy ověřila, že jsme vytvořili plnohodnotný a konkurence schopný produkt.

# 8 Závěr

Cílem tohoto projektu bylo navrhnout systém pro sledování aktivit uživatelů. Prostudovali jsme současný stav aplikací, které se touto problematikou zabývají a především na jejich základě jsme vytvořili požadavky, které by měl splňovat náš systém. Vytvořili jsme architekturu celého systému a namodelovali jsme systém z několika pohledů. Navrhli jsme zpracování událostí na straně agenta, přenos dat z agenta na kolektor, zpracování dat a ukládání do databáze na straně kolektoru a konečně prezentaci výsledků pomocí webové aplikace. Všechny tyto části jsme rovněž implementovali při dodržení všech dosavadních návrhů, omezení a požadavků. Rovněž jsme vytvořili modely databáze a modely a specifikace požadavků celého systému, které celý vývoj značně usnadňují. Neméně důležité bylo i zhodnocení systému z etického pohledu. Prostudovali jsme právní předpisy vztahující se k vyvíjenému systému a zjistili jsme, že v žádném ze zmíněných bodů tento projekt neporušuje zákon.

V projektu jsme se pozastavili i u slabších stránek systému a uvedli jejich možná vylepšení. Mezi tato zlepšení patří především serializace objektů zapisovaných do souboru na straně agenta, která není prováděna příliš efektivně. Dalším krokem možného zlepšení, které jsme diskutovali byla změna protokolu TCP na UDP pro přenos dat mezi agentem a kolektorem. V neposlední řadě jsme navrhli i zlepšení přenosu dat z hlediska jejich bezpečnosti. Všechny zmíněné oblasti, které by bylo možno zlepšit, v žádném případě nemají vliv na funkčnost systému.

Systém byl poměrně dlouhou dobu testován a byl zbaven řady drobných chyb a v jednom případě poměrně závažné chyby. Tato chyba spočívala v neschopnosti agenta získat všechny informace o poměrně důležitých procesech. Problémem bylo především to, že chyba byla velmi obtížně simulovatelná a vyskytovala se jen zřídka. Nakonec, jak jsme si ostatně řekli v kapitole o implementaci agenta, byla i tato chyba odstraněna, a systém by měl být v tuto chvíli bez závažnějších chyb.

Přínosem tohoto systému při správném užití je především jeho schopnost získat údaje o činnosti v síti, a tyto údaje posléze zpracovat takovým způsobem, že z nich vzniknou skutečně důležité informace. Oproti existujícím produktům, věnujícím se též monitorování, je výjimečný v tom, že se nezaměřuje pouze na „špionáž“ pracovníků, ale poskytuje mnohem hodnotnější informace díky zpracování dat o využití softwaru. Je proto schopen usnadňovat rozhodování především při nákupu softwaru, čímž může z finančního hlediska značně firmám ulevit. Ovšem i v oblasti samotného sledování aktivity uživatelů poskytuje prvek navíc, kterým je skutečná aktivita pracovníka v aktivních oknech. Nemůže být proto oklamán pouhým spuštěním pracovní aplikace, ale pracovník

musí skutečně vykonávat nějakou činnost. Jádra agenta a kolektoru navíc byla nasazena v praxi, v softwarovém balíku ALVAO společnosti ALC spol. s r.o. Jedná se o části firmou nazvané „*Software Usage Monitoring*“, tj. využití instalovaného software a „*Personal Audit*“, tj. monitoring uživatelů. Proto lze říci, že celý projekt byl úspěšný a výsledkem je funkční systém pro monitorování uživatelských aktivit.

# Literatura


[1] ARLOW, J. - NEUSTADT, I. UML a unifikovaný proces vývoje aplikací. Brno: Computer Press a.s., 2003. 387 s. ISBN 80-7226-947-X

[2] BODANIS, D. E = $mc^2$ – Životopis nejslavnější rovnice na světě. Přeložil Jan Placht. Praha: Dokořán spol. s r. o., 2002. 279 s. ISBN 80-86569-08-X.

[3] SINGH, S. Kniha kódů a šifer. Přeložili Petr Koubský a Dita Eckhardtová. Praha: Dokořán spol. s r.o., 2003. 382 s. ISBN 80-86569-18-7.

[4] SATRAPA, P. - RANDUS, J. LINUX – Internet server. Praha: Neocortex spol. spol. s r.o., 1998. 413 s. ISBN 80-902230-3-6

[5] MITNICK, K. Umění klamu. Gliwice (Polsko): HELION S.A., 2003. 348 s. ISBN 83-7361-210-6

[6] NAGEL, C. - EVJEN, B. - GLYNN, J. - SKINNER, M. - WATSON, K. - JONES, A. C# 2005 Programujeme profesionálně. Brno: Computer Press a.s., 2006. 1398 s. ISBN 80-251-1181-4

[7] SKOPAL, A. Application for scanning users activities, In: Proceedings of the 14th Conference STUDENT EEICT 2008. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií a Fakulta informačních technologií, 2008. s. 241. ISBN 978-80-214-3615-2

[8] PRICE, J. C# programování databází. Praha: Grada Publishing a.s., 2005. 624 s. ISBN 80-247-0982-1

[9] MACDONALD, M. - SZPUSZTA, M. ASP.NET 2.0 a C# tvorba dynamických stránek PROFESIONÁLNĚ. Brno: ZONER software, spol. s r.o., 2006. 1376 s. ISBN 80-86815-38-2

[10] *eDetektiv – Kontrola zaměstnanců.* URL: <http://www.edetektiv.cz/kontrola.html> [citováno 6.března 2009]

[11] *Monitorovací systém Dozorce – popis programu.* URL: <http://www.dozorce.cz/web/popis-monitorovaciho-systemu/podrobny-popis-programu.php> [citováno 6.března 2009]

[12] *Program pro dokonalé a utajené sledování počítače.* URL: <http://www.spying.cz/> [citováno 6.března 2009]

[13] *Trestní zákon č. 140/1961 Sb. v platném znění, částka 65/1961* URL: <http://aplikace.mvcr.cz/archiv2008/sbirka/1961/sb65-61.pdf> [citováno 26.ledna 2009]

[14] *Trestní zákon č. 262/2006 Sb. v platném znění, částka 85/2006*
URL:<http://aplikace.mvcr.cz/archiv2008/sbirka/2006/sb084-06.pdf> [citováno 26.ledna 2009]

# Seznam příloh

# Příloha 1 – Dotazy SQL

- Zobrazení aktivních oken pracovníka:

```
SELECT A.iActiveWindowId, A.dActiveWindowFrom, A.dActiveWindowTo,
A.sActiveWindowTitle, A.nActiveWindowUserInActiveMinutes,
C.iComputerId, C.sComputer, C.sComputerAdGuid, P.sProcessProductName,
P.sProcessCompanyName FROM ((tActiveWindow AS A LEFT JOIN tComputer AS C ON
A.liActiveWindowComputerId = C.iComputerId)
LEFT JOIN tProcess AS P ON A.liActiveWindowProcessId = P.iProcessId)
LEFT JOIN tPerson AS Per ON A.liActiveWindowPersonId = Per.iPersonId
WHERE CONVERT(Varchar, A.dActiveWindowFrom, 102) = @ParamActiveWindowFrom
AND Per.iPersonId = @ParamPersonId
```

- Zobrazení procesů pracovníka:

```
SELECT DISTINCT P.iProcessId, P.dProcessFrom, P.dProcessTo,
P.sProcessProductName, P.sProcessCompanyName,
P.sProcessOriginalFileName, C.iComputerId, C.sComputer, C.sComputerAdGuid
FROM tProcess AS P JOIN (tComputer AS C JOIN (tActiveWindow AS A JOIN
tPerson AS Per ON Per.iPersonId = A.liActiveWindowPersonId AND Per.iPersonId =
@ParamPersonId) ON A.liActiveWindowComputerId = C.iComputerId) ON C.iComputerId
= P.liProcessComputerId WHERE CONVERT(Varchar, P.dProcessFrom, 102) =
@ParamProcessFrom
```

- Zobrazení výsledků vyhledávání při úplném vyplnění formuláře:

```
SELECT A.iActiveWindowId, A.dActiveWindowFrom, A.dActiveWindowTo,
A.sActiveWindowTitle, A.nActiveWindowUserInactiveMinutes, Per.iPersonId,
Per.sPerson, P.sProcessProductName, P.sProcessCompanyName, C.iComputerId,
C.sComputer FROM (((tActiveWindow AS A LEFT JOIN tProcess AS P ON
A.liActiveWindowProcessId = P.iProcessId) JOIN tPerson AS Per ON
A.liActiveWindowPersonId = Per.iPersonId) JOIN tComputer AS C
ON A.liActiveWindowComputerId = C.iComputerId) WHERE Per.dPersonRemoved    IS
NULL AND C.dComputerRemoved IS NULL AND (A.sActiveWindowTitle LIKE ('%' +
@ParamNameAppTitle + '%') OR P.sProcessProductName LIKE ('%' +
@ParamNameAppTitle + '%')) AND    C.sComputer LIKE ('%' + @ParamComputer + '%')
AND A.dActiveWindowFrom >= @ParamActiveWindowFrom AND
A.dActiveWindowFrom <= @ParamActiveWindowTo AND
A.nActiveWindowUserInactiveMinutes >= @ParamActiveWindowUserInactiveMinutes
```

- Zobrazení výsledků využití software (všichni pracovníci):

```
SELECT DISTINCT P.sProcessProductName, P.sProcessCompanyName,
P.sProcessOriginalFileName, SUM(DISTINCT DATEDIFF(second, P.dProcessFrom,
P.dProcessTo)) / 60 AS iProcessActivitySeconds, SUM(DATEDIFF(second,
A.dActiveWindowFrom, A.dActiveWindowTo)) / 60 AS iWindowActivitySeconds,
SUM(A.nActiveWindowUserInactiveMinutes) AS nInactivityWindowMinutes, Per.sPerson
FROM ((tPerson AS Per JOIN tActiveWindow AS A ON Per.iPersonId =
A.liActiveWindowPersonId) RIGHT JOIN tProcess AS P ON A.liActiveWindowProcessId =
P.iProcessId) WHERE Per.dPersonRemoved IS NULL AND Per.iPersonId =
@ParamPersonId AND(P.dProcessTo IS NOT NULL OR
A.nActiveWindowUserInactiveMinutes IS NOT NULL) AND (P.sProcessProductName
LIKE ('%' + @ParamNameApp + '%') OR P.sProcessCompanyName LIKE ('%' +
@ParamNameApp + '%')) AND P.dProcessFrom >= @ParamProcessFrom AND
(P.dProcessFrom <= @ParamProcessTo OR P.dProcessTo IS NULL)
GROUP BY P.sProcessProductName, P.sProcessOriginalFileName,
P.sProcessCompanyName, Per.sPerson
```

- Zobrazení výsledků využití software (konkrétní pracovník):

```
SELECT DISTINCT P.sProcessProductName, P.sProcessCompanyName,
P.sProcessOriginalFileName, SUM(DISTINCT DATEDIFF(second, P.dProcessFrom,
P.dProcessTo)) / 60 AS iProcessActivitySeconds, SUM(DATEDIFF(second,
A.dActiveWindowFrom, A.dActiveWindowTo)) / 60 AS iWindowActivitySeconds,
SUM(A.nActiveWindowUserInactiveMinutes) AS nInactivityWindowMinutes
FROM tProcess AS P LEFT JOIN tActiveWindow AS A ON P.iProcessId =
A.liActiveWindowProcessId WHERE (P.dProcessTo IS NOT NULL OR
A.nActiveWindowUserInactiveMinutes IS NOT NULL) AND (P.sProcessProductName
LIKE ('%' + @ParamNameApp + '%') OR P.sProcessCompanyName LIKE ('%' +
@ParamNameApp + '%')) AND P.dProcessFrom >= @ParamProcessFrom AND
(P.dProcessFrom <= @ParamProcessTo OR P.dProcessTo IS NULL)
GROUP BY P.sProcessProductName, P.sProcessOriginalFileName,
P.sProcessCompanyName
```

## Příloha 2 - Podrobný popis některých souborů prezentační části systému

- **Web.Config** – tento soubor obsahuje připojovací řetězec, tzv. „*ConnectionString*“, který slouží ke spojení s databází. V souboru je možné provést řadu nastavení, ale z těch důležitých, které jsme použili, jde zejména o autentizační mód, který je nastaven na integrované přihlášení do Windows. To znamená, že pokud aplikaci spustíme, tak login pro přihlášení do operačního systému Windows XP musí být shodný s daty uloženými v databázi.

- **Global.asax** – soubor obsahuje řadu metod, které slouží jako reakce na určité události. Tento soubor je automaticky generován a my pouze vyplníme těla metod, která chceme použít. V našem případě jsme implementovali tělo jediné metody, která se zavolá na událost začátku session. V této metodě nastavíme aktuálně přihlášeného administrátora. Nejprve zjistíme, zda můžeme uživatele autentizovat a připojíme se k serveru SQL, ze kterého se získá jméno uživatele a jeho identifikátor. Identifikátor je posléze uložen do session a slouží k tomu, abychom mohli zjistit, kdo je právě přihlášen a zobrazit jeho jméno.

- **MonitoringPresentationLayerLib.cs** – obsahem tohoto souboru je řada statických metod. Úkolem těchto metod je ověření správnosti URL adresy, nastavení některých hodnot ve stránce, kontrole statusu, zda je uživatel administrátorem a poslední metoda této třídy kontroluje, zda počítač či osoba nemá vazbu na data (okna a procesy). Ověření správnosti URL adresy znamená, že metoda kontroluje, zda uživatel nepřepsal GET parametr. Proto se musí metoda připojit k databázi a dotazem SQL ověřit, zda informace v GET parametru jsou korektní. Pokud zjistí v parametrech chybu, přesměruje uživatele na jinou, obecně dostupnou stránku. Kromě toho, že metoda ověřuje správnost URL adresy, tak ještě zobrazuje na stránce informace získané z databáze. Co se týká ověření statusu administrátora, tak tento krok musíme provádět při mazání uživatelů z databáze. Nesmíme totiž připustit, aby byl smazán poslední administrátor systému. Metoda, která má tuto operaci za úkol, jednoduše spočítá počet uživatelů systému, kteří mají administrátorský status. Poslední metoda nám zkontroluje, zda existuje vazba mezi daty v databázi a stanicí nebo osobou. Důvodem, proč to musíme ověřit, je opět mazání těchto osob nebo stanic, po kterých by mohla v databázi zůstat data, která by posléze postrádala vazbu na stanici či osobu.

- **CurrentUser.cs** – soubor obsahuje dvě metody, kde první slouží pro uložení přihlášeného pracovníka do session a druhá pro jeho získání ze session.

- **MasterPage.master** – jde o soubor, který je hlavní stránkou používanou ve všech ostatních souborech. Jinak řečeno uděláme si vzhled našich stránek v „*MasterPage.master*“, a pak už jen generujeme jejich obsah. Ve všech ostatních stránkách jen oznámíme IIS serveru, že vzhled má vzít z této stránky.

- **ListOfDays.aspx** – jde o zobrazení dní, ve kterých existují nějaké záznamy ke sledovanému pracovníkovi. V souboru „*PersonsActivities.aspx*“ vybereme pracovníka a dostáváme se k tomuto souboru, kde musíme zvolit, který pracovníkův den chceme zkontrolovat. Mnoho stránek komunikujících s databází je založeno na tvorbě dotazů SQL pomocí tzv. SQL datového zdroje. To je nástroj, který nám umožňuje tvořit dotazy SQL přímo ze stránky, tzn. nemusíme se zabývat tvorbou dotazu v asociovaném souboru „*ListOfDays.aspx.cs*“ – z pohledu architektury jde o „*controller*“. Tyto datové zdroje mají velmi dobrou podporu na tvorbu dotazů pomocí dotazovacích editorů. V GET parametru máme identifikátor pracovníka, který snadno získáme ve zmíněném editoru a použijeme ho jako parametr ve vytvořeném dotazu SQL. Tento dotaz zobrazuje datum na základě spojení mezi aktivními okny a pracovníkem. Jinak řečeno vezme se vždy datum, kde je první aktivní okno toho dne. Dále se vytvoří tabulka, která se v ASP.NET nazývá mřížka. Tento prvek obsahuje rovněž řadu prostředků pro usnadnění práce, mezi které mimo jiné patří i rozhraní pro výběr zobrazovaných sloupců. V tomto rozhraní vybereme nejen nadpisy sloupců, ale také obsah buněk. V tomto případě jsou obsahem hypertextové odkazy na zobrazení aktivní oken (soubor „*ActivitiesOfDayWindow.aspx*“) anebo na zobrazení spouštěných procesů (soubor „*ActivitiesOfDayProcess.aspx*“). Odkazy jsou založeny na tom, že obsahují GET parametr s identifikátorem uživatele a datum, které určuje zobrazovaný den.

- **ActivitiesOfDayWindow.aspx** – jak již bylo řečeno, zde zobrazujeme aktivní okna pracovníka pro daný den podle údajů zjištěných v GET parametru. Získávání dat je opět založeno na datovém zdroji a v asociovaném souboru („*controller*“) se pouze nastaví nadpis stránky. Výstup je prováděn do mřížky, která vypisuje řadu důležitých informací. Tyto informace jsou získány na základě téměř všech tabulek v databázi, a proto je dotaz SQL založen na spojení těchto tabulek pomocí operace LEFT JOIN a porovnání atributu začátku

okna s datem přijatým v GET parametru. Stejným způsobem je s GET parametrem porovnán i identifikátor pracovníka. Tento dotaz SQL můžeme vidět jako první v příloze jedna. Každý řádek tabulky obsahuje opět hypertextový odkaz, který směřuje na stránku „*DetailsOfWindow.aspx*“ sloužící k zobrazení podrobností o okně. Tento soubor je tvořen ovládacím prvkem, probraným v předchozí kapitole.

- **ActivitiesOfDayProcess.aspx** – tento soubor je velmi podobný předchozímu, a proto si popíšeme pouze rozdíly mezi těmito soubory. Hodnoty získané v GET parametru jsou stejné, ale co není stejné, je dotaz SQL. V tomto případě skládáme tabulky pomocí operace JOIN. Dotaz začínáme tvořit od tabulky procesů, a navíc výběr musíme omezit klíčovým slovem DISTINCT. Tento dotaz je opět ke zhlédnutí v příloze jedna a je druhý v pořadí. Pokud chceme vidět detaily procesu, dostaneme se pomocí hypertextového odkazu na stránku „*DetailsOfProcess.aspx*“.

- **Searching.aspx** – jedná se o formulář, který slouží k vyhledávání aktivních oken. Na rozdíl od předchozích souborů, je tento převážně založen na souboru „*Searching.aspx.cs*“. Jeho úkolem je zkontrolovat zadané hodnoty a v případě, že jsou v pořádku nás přesměruje na „*ResultOfSearching.aspx*“. „*Controller*“ musí ověřit, zda je datum, které uživatel zadal, správně zadáno. Toto ověření provádí pomocí syntaktické analýzy zadaného data, které může v případě chyby skončit výjimkou. Na stejném principu je založena kontrola hodnoty neaktivity osoby. V GET parametru potom předává název nebo nadpis aplikace, název počítače, datum a čas, podle kterého se hledá, která okna se stala aktivní v této nebo v pozdější době, a konečně datum a čas, podle kterého se hledá, která okna se stala aktivní v této nebo před touto dobou. Posledním předávaným GET parametrem je doba, podle které se vyhledávají pouze okna, ve kterých byl pracovník tuto nebo delší dobu neaktivní. Tyto GET parametry samozřejmě lze kombinovat různým nastavením formuláře.

- **ResultOfSearching.aspx** – jde o soubor pro zobrazení vyhledaných výsledků. Pomocí datového zdroje si soubor načte GET parametry, kde posléze v souboru „*ResultOfSearching.aspx.cs*“ zjišťuje, které z nich byly skutečně poslány. Nejprve se ale provede kontrola, zda nebyl některý parametr ručně přepsán, a to stejným způsobem jako v souboru „*Searching.aspx*“. Pak se vytvoří základ dotazu SQL, a podle poslaných GET parametrů se formuje zbytek dotazu. Základem dotazu je výběr zobrazovaných informací,

kde jsou opět hojně využity operace JOIN a LEFT JOIN. Tento dotaz se vytváří do proměnné v souboru, který je pro nás „*controller*“. Vidíme proto, že v tomto případě dotaz nevytváříme pomocí editoru, ale všechnu práci plní náš „*controller*“. Zajímavé jsou především části dotazu, ve kterých figuruje název nebo nadpis okna. Pomocí operátoru LIKE totiž můžeme zadat pouze část hledaného řetězce. Na stejném principu se vyhledává i podle stanice, tj. Stačí pouze část názvu stanice. Další parametry určují interval, ve kterém se mohlo okno stát aktivním. Poslední částí dotazu můžeme zadat pracovníkovu neaktivitu v okně. Tento dotaz můžeme zhlédnout jako třetí v pořadí v první příloze.

- **SwUsing.aspx** – tento formulář slouží k vyhodnocení používání softwaru na stanicích. Opět obsahuje datový zdroj, který má ovšem zcela jiný význam než v předchozích případech. Tento slouží k tomu, aby načetl seznam pracovníků do posuvného seznamu. Zbytek vyhledávání se odehrává v asociovaném souboru „*SwUsing.aspx.cs*“. Tento soubor slouží ke kontrole zadaných hodnot a k přesměrování s potřebnými GET parametry na stránku „*ResultOfUsingSw.aspx*“. Jakmile uživatel stiskne tlačítko, „*controller*“ to zaznamená a zjistí, zda je zadán název aplikace. Pokud ano, přidá jej k odkazu jako GET parametr. Dále kontroluje, zda jsou zadány data, a zda jsou ve správném formátu, a když ano, opět je přidá do GET parametru. Nakonec přidá do GET parametru identifikátor pracovníka. Pokud jsou vybráni všichni, tak v takovém případě vloží do GET parametru nulu. Je nezbytné vědět, že v tomto případě jde o vyhledávání aplikací podle jejich názvu, takže zobrazovat se budou i procesy, které vůbec uživatelské rozhraní nemají. Další výjimečná situace nastane, pokud ve formuláři nevyplníme nic. I v takovém případě poskytne soubor informace, a to sice výpis všech nalezených programů na všech počítačích.

- **ResultOfUsingSw.aspx** – tento soubor je založen na získání GET parametrů pomocí datového zdroje a zbytek se odehrává v souboru „*ResultOfUsingSw.aspx.cs*“. Tento „*controller*“ nejprve znovu zkontroluje správnost GET parametrů, a pokud zjistí, že v parametru „*iPersonId*“ (identifikátor pracovníka) je něco jiného než nula, dotáže se na jméno pracovníka a zobrazí jeho jméno na stránce. Následně začneme tvořit dotaz SQL pro zobrazení všech potřebných informací. Základ dotazu závisí na tom, zda byl zadán identifikátor konkrétního pracovníka a pokud ano, omezíme informace pouze na tohoto pracovníka. Kromě obecných informací zobrazujeme především tři důležité časové položky. Za prvé jde o celkovou dobu běhu aplikace, která se získává z doby běhu procesů. Rozdíl

časových hodnot musíme získávat ve vteřinách, a až teprve potom přepočítat na minuty, protože při sledování minut by došlo ke značnému zkreslení výsledků. Druhou hodnotou je celková doba běhu aplikace na popředí, která se získává z doby aktivity jednotlivých oken. Poslední hodnotou je celková doba neaktivit v aplikaci. Celý dotaz v obou verzích (pro konkrétního pracovníka i pro všechny pracovníky dohromady) je uveden jako čtvrtý a pátý v pořadí, v první příloze.

- **AdminDatabase.aspx** – tento soubor slouží pouze ke zobrazení lokálního menu pomocí tří tlačítek „*Mazání záznamů*“, „*Správa osob*“ a „*Správa počítačů*“. V tomto pořadí si probereme i související soubory.

- **AdminDeleteItems.aspx** – jde o formulář sloužící k nastavení mazaných položek. Můžeme vybrat, zda chceme mazat podle počítačů nebo podle pracovníků. K tomuto účelu slouží zatrhovací pole, po jehož zatržení nebo odtržení se vyvolá metoda, která provede načtení seznamu pracovníků nebo počítačů do posuvného seznamu. Samotné mazání probíhá tak, že kolektor si nejprve zjistí jakým způsobem se bude mazat. Možnosti jsou celkem čtyři. Mohou se mazat záznamy konkrétního pracovníka, konkrétního počítače, všech pracovníků anebo všech počítačů. Dále se zkontroluje správnost zadání obou dat. Pokud se zjistí, že se mají mazat informace podle všech stanic nebo pracovníků, tak dotaz SQL je pro obě možnosti stejný a není ničím omezen. Nejprve se smažou procesy a hned po nich zbývající okna. V případě, že došlo k výběru mazání podle konkrétní stanice, tak je situace také celkem jednoduchá. V obou dotazech (u mazání procesů a u mazání oken) se pouze přidá podmínka vazby na identifikátor vybrané stanice. Nejsložitější je mazání záznamů podle vybrané osoby. Toto mazání se musí provádět ze strany oken, protože v databázi neexistuje vazba mezi pracovníky a procesy. Proto je nutné nejprve najít okna vázaná k tomuto pracovníkovi. Na základě těchto oken se najdou procesy, které se odstraní a hned po nich se odstraní i okna. Po smazání následuje přesměrování na stránku „*ResultDeleting.aspx*“ i s GET parametry.

- **ResultDeleting.aspx** – tato stránka slouží pouze pro zobrazení výsledků mazání. Z GET parametrů se zjistí počty smazaných položek a ty se oznámí uživateli, který mazání inicioval.

- **AdminPersonDetails.aspx** – k zobrazení této stránky může dojít dvěma způsoby. Stránka „*AdminPerson.aspx*“ probíraná v podkapitole o ovládacích prvcích obsahuje tlačítko „Přidat

osobu“, po jehož stisku se na tuto stránku přesuneme. Anebo jsme na stránku přesměrováni ve chvíli, kdy chceme zobrazit detaily již existujícího pracovníka. Rozdíl je v tom, že v prvním případě se na tuto stránku dostáváme s hodnotou GET parametru „*iPersonId*“ (identifikátor pracovníka) nula, zatímco při zobrazení detailů existujícího uživatele je hodnota tohoto parametru rovna identifikátoru pracovníka v databázi. Základem této stránky je prvek zvaný náhled detailů, který se zobrazuje v několika módech. Mód se určuje právě podle hodnoty GET parametru „*iPersonId*“. Má-li tento parametr hodnotu nula, tak víme, že budeme přidávat nového uživatele. V takovém případě se náhled detailů zobrazí prázdný s možností zapisovat nové hodnoty a potvrdit je stiskem tlačítka „Vložit osobu“. Základem komunikace s databází není „*controller*“ AdminPersonDetails.aspx.cs, ale datový zdroj, který obsahuje dotazy jak pro editaci (dotaz UPDATE) a výběr dat (dotaz SELECT) před editací, tak pro vkládání (dotaz INSERT). Pokud jde o vkládání, tak to zajistí sám datový zdroj a „*controller*“ pouze reaguje metodou, která se automaticky zavolá bezprostředně po vložení. V metodě se zjistí identifikátor, který byl osobě v databázi přidělen pomocí: `SELECT @@IDENTITY` a přesměruje uživatele zpět na tuto stránku ovšem s GET parametrem „*iPersonId*“. Hodnota tohoto parametru je právě hodnota získaného identifikátoru. Ve výsledku jsme tedy na stejné stránce, ovšem v módu editace. Tento mód obsahuje dvě tlačítka, a to sice „Editovat“ a „Odstranit osobu“. Při stisku tlačítka „Editovat“ se přepne náhled detailů do editačního módu, kde jsme již schopni zapisovat. Po stisku tlačítka „Uložit změny“ se opět aktivuje datový zdroj, který vyvolá dotaz UPDATE. Těsně před zapsáním změny do databáze, je vyvolána metoda, ve které se kontroluje správné zadání dat. Pokud něco není v pořádku, akce může být stále ještě přerušena. Je-li však vše v pořádku, změny jsou uloženy a mód je přepnut na editační bez možnosti zápisu. Pokud se rozhodneme pracovníka smazat, vše probíhá stejně, jen se navíc kontroluje, zda pracovník není posledním administrátorem, a zda nemá stále vazbu na data. Jeho restaurace se provádí opět v editačním módu, kdy se pouze odstraní datum jeho smazání.

- **AdminComputers.aspx** – tento soubor zobrazuje seznam stanic. V principu jde o naprosto stejnou stránku, jakou je „*AdminPersons.aspx*“, s tím rozdílem, že tato nevyužívá ovládací prvek, protože se v celé aplikaci vyskytuje pouze jedenkrát. Umožňuje vybrat konkrétní stanici ze seznamu a také přidat novou stanici nebo zobrazit detaily stanice existující. Tyto dva případy opět přesměrují na stejnou stránku, a to sice na „*AdminComputerDetails.aspx*“.

- **AdminComputerDetails.aspx** – soubor je obdobou souboru „*AdminPersonDetails.aspx*“, a jelikož by popis tohoto souboru byl stejně rozsáhlý, tak si pouze povíme o odlišnosti vzhledem k tomuto souboru. Jediný rozdíl je, že zde při mazání stanice dochází pouze ke kontrole, zda má vazbu na data, zatímco u osob jsme byli nuceni ještě kontrolovat, zda není osoba posledním administrátorem v systému.