Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Vzdálená správa Java aplikací z medicínského prostredí

Plzeň 2014 Lubomír PETERA

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 29. dubna 2014

Lubomír PETERA

Abstract

Distant management of medical Java applications

This bachelor thesis focuses on management and monitoring of the medial applications developed in Java.

The introduction of the theoretical part provides a basic outline of the possibility of application monitoring. This is followed by a summary describing the specific software product - medical software Medical Process Assistant. At the end of the theoretical part there is a synopsis of Java management extensions technology.

The practical part of this thesis deals with the description of the software solution, which was designed, implemented and nowadays it is used for monitoring as well as management of the product Medical Process Assistant.

Obsah

1 Úvod

V dnešní době už neplatí, že chce-li člověk napsat nějaký program, musí si nejdříve spájet programovatelnou desku, navrhnout programovací jazyk, napsat překladač a poté se může pustit do programování. Typická úloha moderního programátora je založena na znalosti rozhraní dvou vrstev, mezi které píše svůj program. Programátor obvykle nemá detailní znalost celého systému.

Systém jedné či více komunikujících aplikací je složen z několika stovek takto naprogramovaných úseků. To ovšem znamená velká rizika v případě programátorovy chyby. Bohužel bezchybný program je utopická myšlenka a každý program nějaké chyby obsahuje. Tyto chyby se pak vlivem neznalosti nižších vrstev aplikace mohou zvětšovat do katastrofických rozměrů.

Proto je v dnešní době nezbytnou součástí vývoje softwaru testování. Testování je náročný a zdlouhavý proces, který dokáže odhalit většinu chyb, ale nikdy nedokáže podchytit všechny možné problémy. Další úskalí testování je simulace podmínek, ve kterých budou aplikace běžet v ostrém provozu. To také není jednoduché, ba je to prakticky nemožné (nelze vytvořit přesnou kopii produkčního prostředí pro testovací účely).

Aplikace se středně kritickou důležitostí, jakou bezesporu nemocniční informační systém je, musí podporovat ještě další úroveň zabezpečení proti programovým chybám a tou je možnost sledování a správy běžících aplikací. Příkladem může být sledování obsahu vyrovnávací paměti, nebo nastavení proměných běhového prostředí.

Cílem této práce je navrhnout nástroj pro sledování a správu aplikací v rámci balíku lékařských aplikací Medical Process Assistant. Medical Process Assistant je distribuovaný nemocniční software, který je převážně napsaný v programovacím jazyku Java. Je založen na interakci služeb běžících na serverových počítačích a klientských aplikacích běžících na osobních počítačích v rámci nemocniční sítě.

Systém aplikací založený na aplikacích běžících ve virtuálním stroji Javy poskytuje mnoho vestavěných možností, jak lze aplikaci sledovat. Například není problém vytvořit otisk aktuálně alokované paměti procesu - takzvaný MemoryDump. MemoryDump lze použít k analýze stavu aplikace. Avšak

analýza stavu procesu tímto způsobem není triviální a je značně zdlouhavá.

Navržené řešení by tedy mělo být co nejjednodušší na použití. Mělo by poskytovat aktuální informace o stavu procesu. A mělo by dovolit s tímto stavem do jisté míry manipulovat, aby bylo možné případné problémy řešit. V ideálním případě by se mělo problémům předcházet.

Požadavek na systém správy aplikací je, aby byl založen na technologii Java management extensions. Technologie Java management extensions je standardní způsob, kterým lze získávat informace o běžícím virtuálním stroji Javy a na požádání z vnějšku virtuálního stroje v něm spouštět libovolný, předem definovaný kód.

2 Teoretická část

První část této práce je rozdělena do tří celků:

- Stručný přehled možností sledování aplikací
- Popis softwarového řešení Medical Process Assistant
- Přehled technologie Java management extensions

Java aplikace lze, jako každé jiné aplikace, sledovat standardními nástroji operačního systému. Avšak Java aplikace poskytují pokročilé možnosti sledování a následného spravování aplikace a virtuálního stroje, na kterém aplikace běží.

Medical Process Assistant je softwarové řešení realizované převážně v programovacím jazyku Java. Tento software je založen na interakci mezi klientskou aplikací a serverovou službou. Serverové služby odstraňují nutnost komunikace klientské aplikace přímo s databází a poskytují jednotné rozhraní dat bez ohledu na konkrétní typ databáze.

Klientské aplikace systému Medical Process Assistant jsou plnohodnotné Java aplikace. Díky tomu, že se nejedná o webové aplikace, které by byly spouštěny převážně na straně serveru, ale o aplikace běžící převážně na straně klienta, je možné část výpočetního výkonu využít na klientské straně.

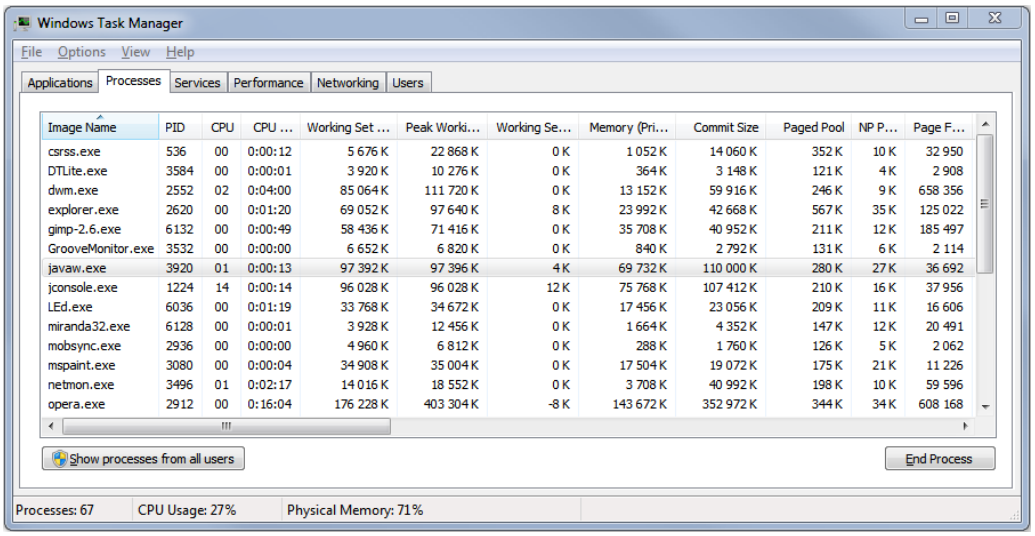
Vlastní sledování a následné spravování bylo realizováno dle specifikace Java technologie Java management extensions. Tato technologie je standardní způsob sledování a správy Java aplikací a její podstatná část je integrována do standardních knihovnen Javy.

Technologie Java management extensions je rozdělena do tří vrstev, což umožňuje značnou modularitu a jednodušší implementaci konkrétního řešení.

Hlavní stavební kameny technologie Java management extensions jsou takzvané MBeany, kterých existuje několik druhů. Nejjednodušší implementace MBeanu je java třída implementující specifické rozhraní.

2.1 Možnosti sledování Java aplikací

Java aplikace lze, jako každé normální aplikace běžící v operačním systému, sledovat nástroji tohoto operačního systému. V rámci operačního systému Windows je standardně dodáván nástroj Windows Task Manager, který lze spustit stiskem kláves Ctrl+Shift+Esc. Spuštěný Windows Task Manager má přehledné prostředí - viz obrázek 2.1.

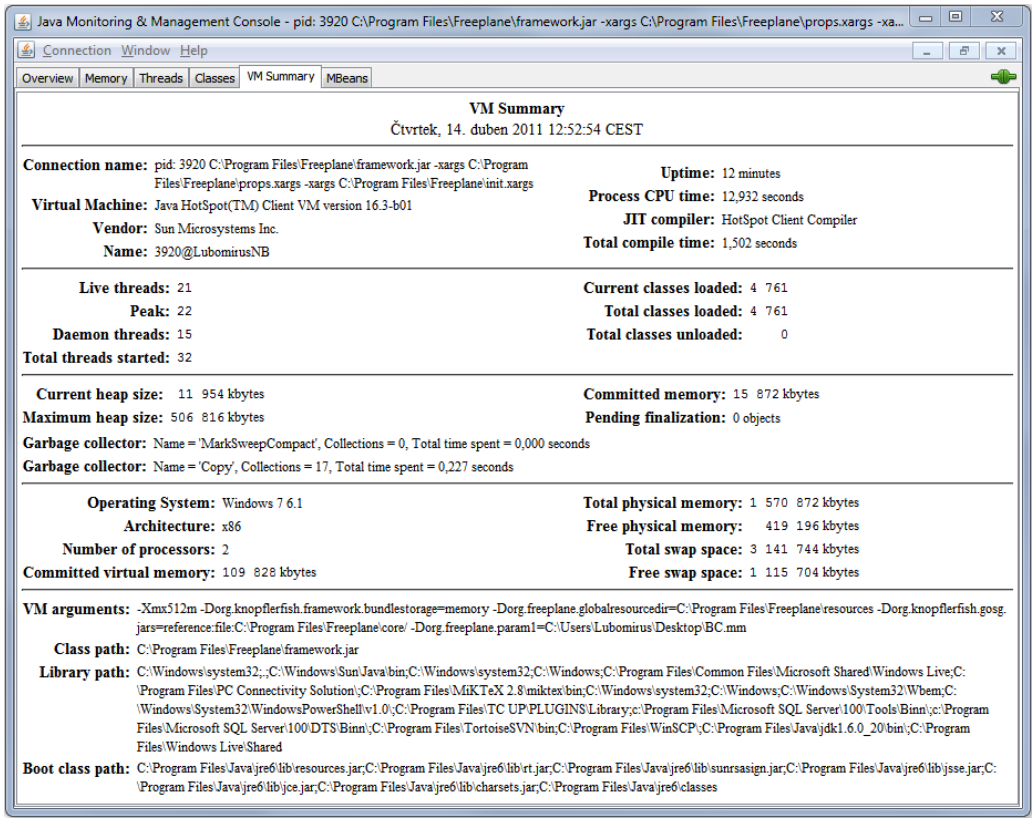


Obrázek 2.1: Windows Task Manager

Informace, které může poskytnout Windows Task Manager:

- Identifikace procesu - proces id (PID)
- Velikost alokované paměti
- Počet vláken procesu
- Informace o IO operacích
- Počet přidělených referencí na soubory
- Parametry, se kterými byl proces spuštěn
- Identifikace jádra procesoru, na kterém proces běží

Systémoví správci nabízí pouze základní obecné informace o spuštěných procesech z pohledu operačního systému. O spuštěné Java aplikaci lze však získat daleko více informací. Pokročilé možnosti sledování nabízí specializovaný nástroj JConsole dodávaný standardně spolu s Java vývojářským balíkem JDK.



Obrázek 2.2: JConsole

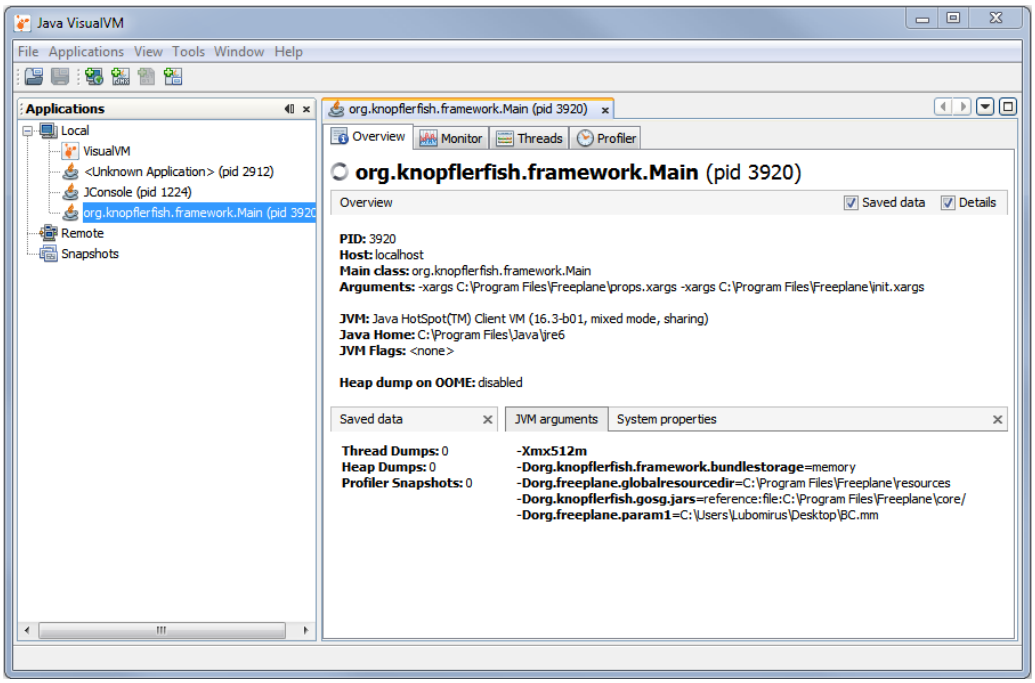
Na obrázku 2.2 je zobrazen spuštěný program JConsole připojený k Java aplikaci. JConsole právě zobrazuje informace o běžícím virtuálním stroji Javy. Zobrazené informace v JConsoly jsou obsáhlejší než informace systémového správce. Navíc obsahují informace specifické pro Java aplikace. Mezi další informace patří například:

- Verze virtuálního stroje
- Lokace class souborů = Class path
- Počet vláken dle jejich typů

- Počet nahraných tříd
- Velikost alokované paměti a její rozdělení do jednotlivých oblastí
- Přehled volaných metod pro jednotlivá vlákna - Stack trace vláken
- Detekce uvíznutí vláken - Deadlock

Nástroj JConsole dovoluje aplikaci nejen monitorovat, ale také částečně spravovat prostřednictvím technologie Java management extensions. JConsole slouží jako standardní Java management extensions klient, a tak dovoluje například požádat běžící virtuální stroj o spuštění garbage collectoru.

Podobným nástrojem pro sledování Java aplikací je VisualVM, který je taktéž dodáván v Java vývojářském balíku JDK. Z hlediska sledování virtuálního stroje nabízí podobné možnosti jako JConsole. Navíc přidává možnost vytvořit otisk alokované paměti programu - Heap dump. A umožňuje prakticky plnohodnotné profilování aplikace. Prostředí VisualVM je vidět na obrázku 2.3 .



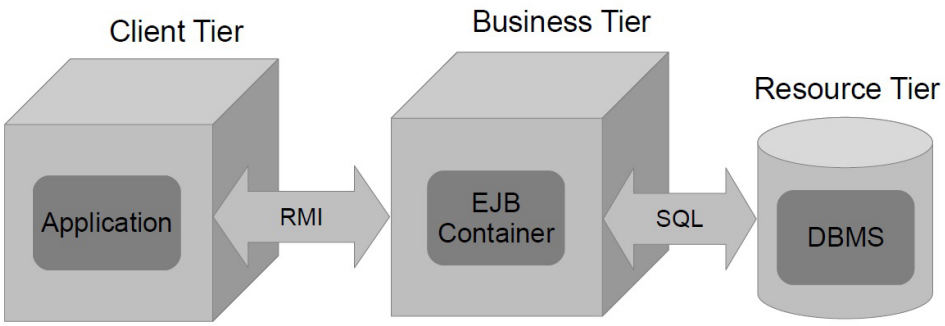
Obrázek 2.3: VisualVM

2.2 Medical Process Assistant

Medical Process Assistant (zkráceně MPA) je softwarové řešení vyvíjené rakouskou firmou Systema Human Information Systems GmbH [Sys(2011)]. Jedná se o nemocniční informační software mapující medicínské procesy s důrazem na modularitu systému a snadné přizpůsobení jakémukoliv procesu, který lze v nemocničním prostředí mapovat. Mezi procesy, které lze prostřednictvím MPA mapovat a plánovat, patří například:

- Management nemocničních lůžek
- Plánování operací
- Správa databáze pacientů
- Mapování návštěv pacientů v ordinacích
- Seznam úkolů na nemocničním oddělení

Vývoj MPA začal před deseti lety v Rakousku. Záhy byl software nasazen a dodnes je instalován a používán v několika stovkách nemocnic v několika zemích. Na vývoji se nyní podílí mezinárodní distribuovaný tým, jehož část je i v České republice. Stále jsou vydávány nové verze, a to jak záplaty chyb, tak verze s novými funkcemi.



Obrázek 2.4: Model Rich client application, zdroj: [TRONÍČEK(2011)]

MPA je distribuovanou aplikací založenou na architektuře klient server. Dle knihy [Ajay D. Kshemkalyani(2011)] lze MPA zařadit mezi tipickou aplikace Rich client application neboli aplikace s plnohodnotným klientem. Strukturu archetypu popisuje obrázek 2.4 .

Hlavní výhody a nevýhody tohoto typu aplikací uvedené v přednášce shrnuje tabulka 2.1. Na rozdíl od aplikace, která by byla založena na modelu Web application (respektive Rich Internet application), je velká část výpočetního výkonu potřebného k běhu aplikace konzumovaná na straně klienta, a tak je výrazně ulehčeno serverovému hardwaru.

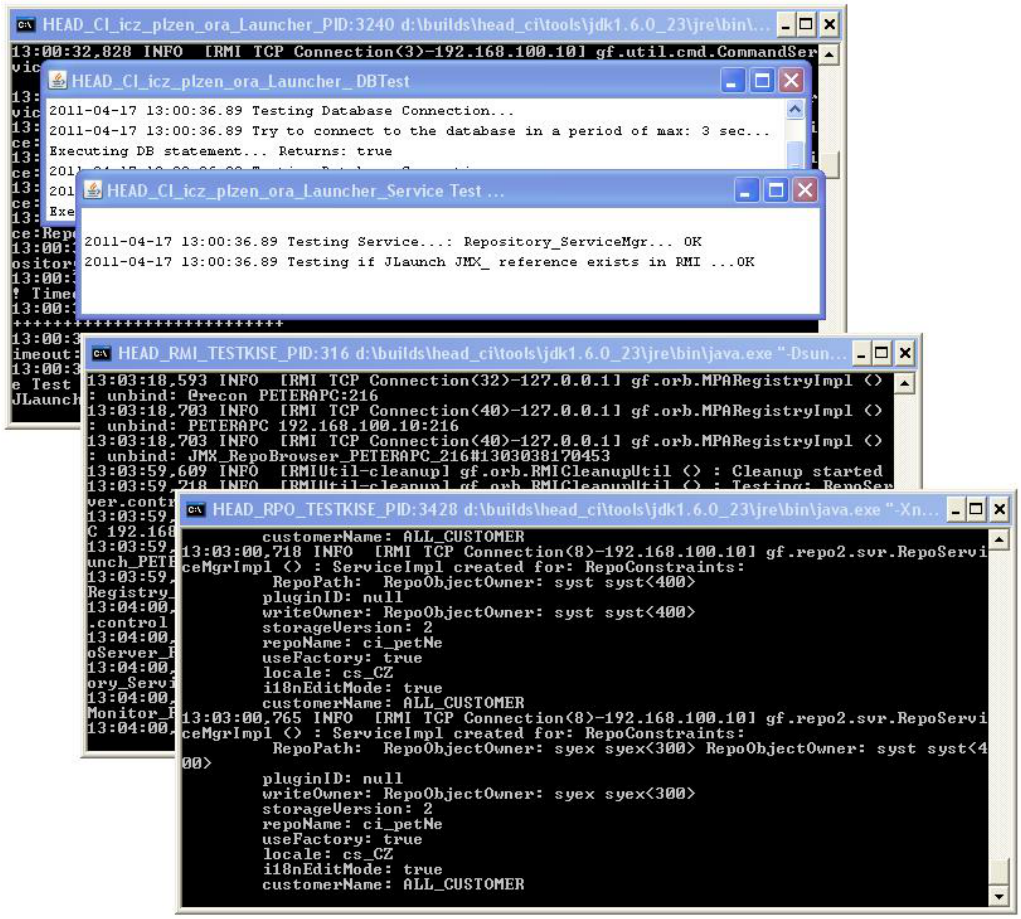
Výhody	Nevýhody
rich UI	upgrade
interactive UI	management
offline support*	

Tabulka 2.1: Výhody a nevýhody modelu Rich client application

Z výhod uvedených v tabulce 2.1 neplatí pro MPA offline support, čili možnost pracovat s aplikací i bez komunikace se serverem. Tato možnost není v MPA plnohodnotně uplatněna. Komunikace se serverovými službami je zapotřebí po celou dobu běhu klientských aplikací. Pokud je komunikace se serverovou službou přerušena například z důvodu síťových problémů, nebo restartování serverové služby, přechází klientská aplikace do stavu čekání na obnovení spojení se serverovou službou. Po opětovném navázání spojení aplikace přejde zpět do normálního pracovního režimu. Klientskou aplikaci není zapotřebí restartovat.

2.2.1 Serverové aplikace

Serverové aplikace jsou služby poskytující informace klientským aplikacím. Slouží zejména k ukládání a sdílení informací přicházejících od klientů. Dále serverové služby odstiňují klienty od konkrétního úložiště dat. V případě, že se změní databázový program, klientský program by tuto změnu neměl pocítit; služba poskytující informace z databázového systému vrací klientovi data ve stejné podobě bez ohledu na druh databáze. Služby dále slouží k zachování informací a tím ulehčují provoz databázi. Pokud si několik klientů požádá o stejná data, není zapotřebí opakovat dotaz do databáze. Výsledek prvního dotazu je uchován a je vracen dalším klientům. Není zapotřebí zatěžovat databázi stejnými dotazy. Spuštěné služby je vidět na obrázku 2.5.



Obrázek 2.5: Služby

O spuštění, monitorování běhu a zastavení služeb na serveru se stará služba JLaunch. Tato služba má definovány závislosti mezi službami a pomocí jednoduchého konfiguračního souboru umí spustit služby v korektním pořadí.

Po spuštění poslední služby je dále ve službě JLaunch spuštěno vlákno ServiceChecker, které kontroluje, zdali jsou všechny služby spuštěné. Toto vlákno nekontroluje jejich stav, pouze zdali daná služba běží. V případě, že některá služba z listu služeb neběží, je ihned spuštěna znovu.

Další vlákno, které se spouští po startu všech aplikací je DBChecker. Toto vlákno kontroluje, zdali je možné spojit se s databází. Vlákno posílá jednoduchý sql dotaz a očekává odpověď. V případě, že se tato zkouška několikrát nepovede, přechází vlákno do stavu ztráty kontaktu s databází. Ve stavu ztráta kontaktu s databází vlákno postupně požádá všechny služby o ukon-

čení. Po obnovení spojení s databází vlákno opět spustí všechny služby. Tento postup zamezuje zahlcení databáze po jejím restartu a předchází problémům s poškozenými službami vlivem absence spojení s databází.

Obě vlákna - ServiceChecker a DBChecker, fungují na bázi periodických dotazů. Periodu dotazů lze nastavit parametry příkazové řádky služby JLaunch. Standardní nastavení je v řádu desítek dotazů za hodinu. Jako jediná služba má JLaunch pro tyto vlákna jednoduché gui v podobě oken s výpisem událostí (prakticky velice podobné příkazovému řádku). Ostatní služby nemají grafické uživatelské rozhraní.

První službou, kterou musí služba JLaunch spustit, je služba RMIRegistry. Tato služba zprostředkovává navázání komunikace mezi aplikacemi a je tedy nezbytná pro správný běh všech služeb a klientů. Každý spuštěný program, tedy všechny služby i klienti, mají povinnost registrovat se v této službě. Zároveň každý spuštěný program ví, kde má hledat službu RMIRegistry. Služba RMIRegistry má tedy přehled o adresách všech běžících služeb a klientů. Potřebuje-li klient navázat komunikaci se službou požádá službu RMIRegistry, aby poskytla adresu dané služby. Klient tedy musí znát pouze adresu služby RMIRegistry a identifikaci služby, se kterou chce navázat spojení. Toto má jistě nesporné výhody, mimo jiné to umožňuje flexibilní rozložení služeb a klientů v rámci sítě.

Služba RMIRegistry periodicky kontroluje reference na aplikace a odstraňuje neplatné reference na neběžící aplikace. Tento proces je spouštěn typicky několikrát za minutu. Ve skutečnosti vlákno ServiceChecker služby JLaunch kontroluje pouze, zdali jsou ve službě RMIRegistry reference na všechny sledované služby. Služba JLaunch je výjimkou v registraci do služby RMIRegistry. Služba JLaunch se neregistruje při spuštění, ale až po startu služby RMIRegistry.

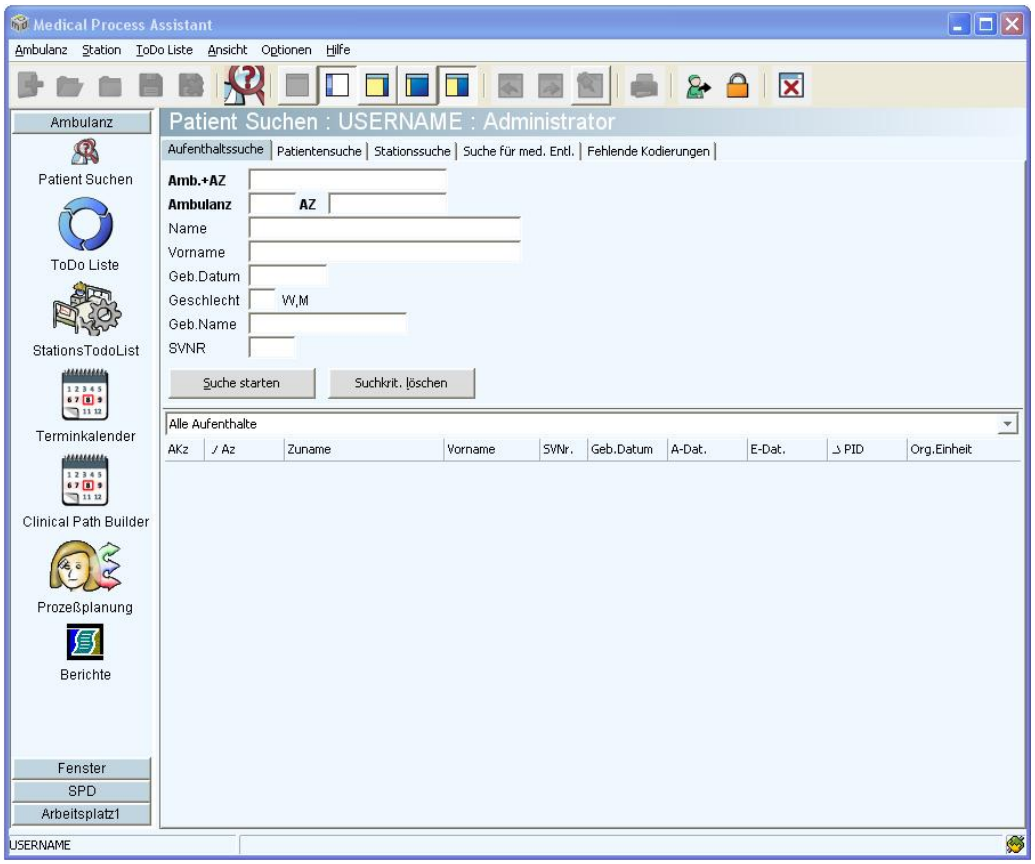
Obvykle druhou spouštěnou službou je služba RepoServer. Tato služba poskytuje ostatním službám a klientům celé Java objekty, které dokáže perzistentně ukládat do úložiště zvaného Repository.

Z hlediska klientů služby RepoServer je nezbytné zažádat managera managera repository objektů o poskytnutí, respektive vytvoření managera repository objektů a tohoto managera poté požádat o daný Java objekt. Z hlediska služby RepoServer služba vrací na dotaz klienta serializovanou instanci Java třídy, která je uložena perzistentně v databázi. V Repository jsou uloženy objekty reprezentující databázovou strukturu tabulek, definice reportů, některé

vlastnosti (properties) aplikací, definice grafického uživatelského rozhraní klientských aplikací a mnoho dalších.

2.2.2 Klientské aplikace

Klientské aplikace zprostředkovávají komunikaci mezi serverovými službami a uživateli. Obecně lze rozdělit klientké aplikace na Working centra a ostatní aplikace. Ostatní klientské aplikace jsou obvykle specifické jednoúčelové programy, jako například prohlížeč Repository RepoBrowser, inicializátor repository, nástroj pro správu internacionalizace a jiné. Mezi ostatní klientské aplikace lze zařadit i automatické testy, které jsou inicializovány jako klientské aplikace.

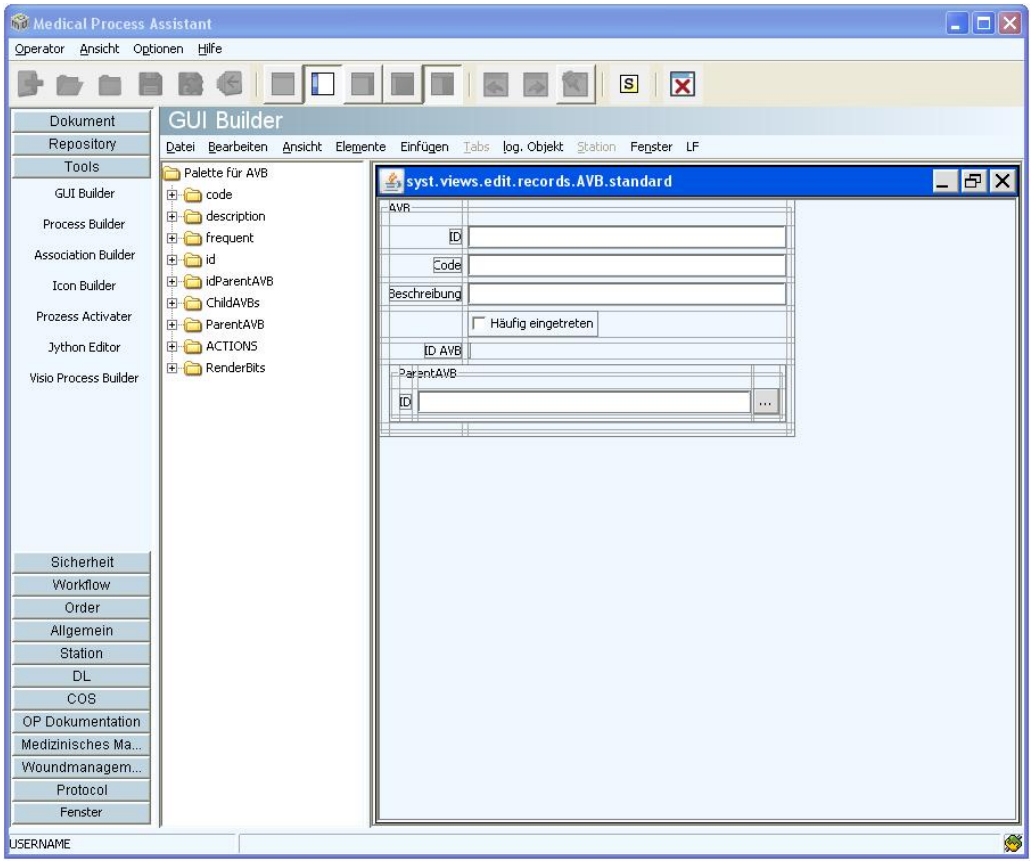


Obrázek 2.6: DefaultWorkBench s německým prostředím

Working centra jsou klientské aplikace, pro jejichž používání je zapotřebí se přihlásit vyplněním uživatelského hesla a jména. Po přihlášení se uživateli načte prostředí programu dle definice která je uložena v Repository.

Mezi základní Working centra patří DefaultWorkBench (na obrázku 2.6). To je základní Working centrum pro přihlašování sester a doktorů. Nachází se zde typicky pracovní prostředí staničních sester.

Druhé nejpoužívanější Working centrum (na obrázku 2.7) je OperatorWorkingCentrum, které slouží zejména ke správě celého systému, vytváření definic procesů, spravování repository a vytváření jiných technických definic.



Obrázek 2.7: OperatorWorkingCentrum s německým prostředím

2.3 Java management extensions

Java management extensions (zkratka JMX) je java technologie, která přidává možnost sdílet informace o běžícím virtuálním stroji java platformy a poskytuje veliké možnosti tento virtuální stroj sledovat a zároveň i spravovat. Poskytování některých informací je již implementováno ve standardní java knihovně. Poskytování dalších specifických informací si lze doimplementovat. Rovněž některé základní prostředky pro ovládání činnosti aplikace jsou již implementovány (například požádání o spuštění garbage collector sledovaného virtuálního stroje). Lze si doimplementovat libovolné další ovládací prvky. Specifikace technologie určuje pouze rozhraní pro tvorbu ovládacích prvků a záleží na implementaci, co bude dělat.

2.3.1 Historie technologie JMX

Technologie Java management extensions, byla jako většina Java technologií, vyvíjena prostřednictvím komunitního procesu Java specifikation request (JSR). První verze, 1.0 , 1.1 a 1.2, byly vyvíjeny pod označením JSR 3 [Sun(2002)] (finální verze vydána 7. září 2000). Verze 2.0 byla vyvíjena jako JSR 255. Dle [Sun(2009)] v současné době však není JSR 255 aktivní a není ani zahrnuto do Javy verze 7 (respektive do openJDK 1.7). Specifikace vzdáleného přístupu k virtuálnímu stroji Javy byla vyvíjena jako JSR 160 [Sun(2003)] a finální verze byla vydána 23. října 2003.

Od Javy verze 5 je JMX součástí standardní knihovny Javy. Což znamená velké usnadnění při nasazení aplikace do provozu. Není zapotřebí instalovat další knihovny, ale postačí standardní instalace Javy.

2.3.2 Architektura technologie JMX

Technologie JMX je založena na třívrstvé architektuře. Tato architektura dovoluje jednodušší práce s jednotlivými vrstvami. Záměna implementace v rámci jedné vrstvy ovlivní pouze minimálně implementaci v sousední vrstvě a prakticky se nedotkne vzdálené vrstvy.

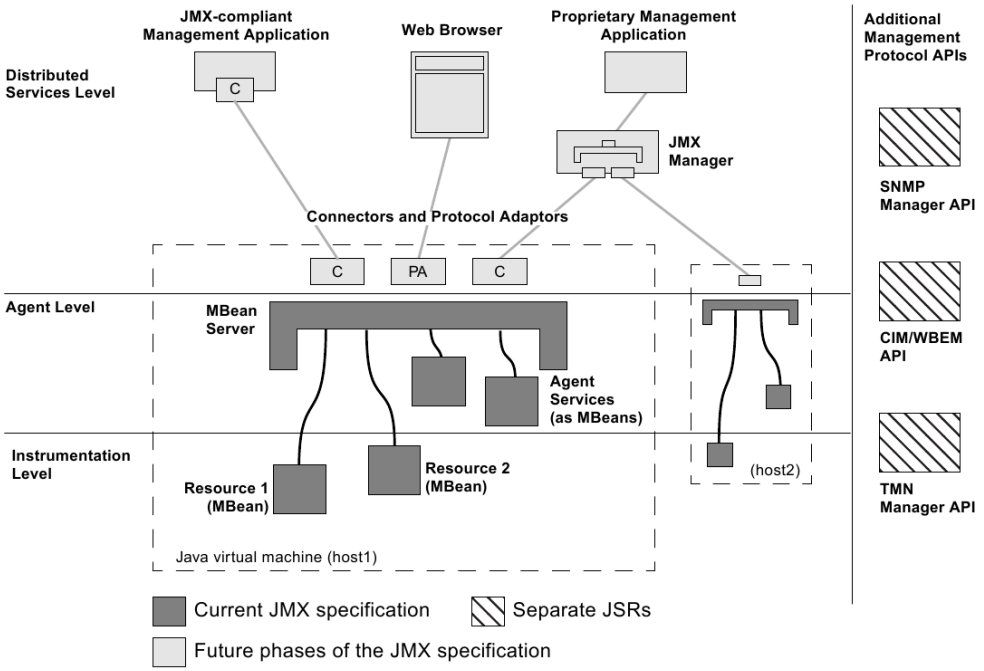
Technologie Java management extensions se dělí na vrstvy:

Instrumentation vrstva Vlastní výkonný kód, který lze spouštět uvnitř virtuálního stroje. Na této úrovni se nachází spravovatelné zdroje. Více o těchto zdrojích je v kapitolách Druhy MBeanů a Notifikace.

Agent vrstva Vrstva zprostředkovávající komunikaci mezi Instrumentation - vnitřní vrstvou a vnějším světem. Více v kapitole Agent Service a MBean server.

Distributed management vrstva Klientská vrstva, která sleduje virtuální stroj zvnějšku.

Rozmístění a spojení vrstev ukazuje obrázek 2.5 .



Obrázek 2.8: Architektura technologie JMX, zdroj: [Sun(2002)]

2.3.3 MBean server

MBean server je registrátor MBeanů ve virtuálním stroji. Nachází se na úrovni Agent vrstvy. Poskytuje informace o registrovaných MBeanech Distributed

management vrstvě. MBean server poskytuje vždy pouze informace o JMX rozhraní MBeanů, nikdy jejich implementaci. MBeany jsou registrovány s unikátním jménem objektu (object name). Toto jméno poté používá klient k přístupu k jednotlivým JMX rozhraním MBeanů.

2.3.4 Agent services

Agent services jsou objekty, které přímo operují s MBeany. Nachází se v Agent vrstvě technologie JMX. Starají se o spouštění operací (metod MBeanů) a o podobné technické věci kolem MBeanů. Specifikace JMX definuje následující Agent services:

Dynamic class loading Dovoluje dynamické načítání tříd i ze vzdálených zdrojů.

Monitors Sleduje hodnoty atributů MBeanů a dovoluje informovat objekty o jejich změnách.

Timers Poskytuje plánovací služby. Může pracovat na bázi jednorázového spuštění, či opakovaného - periodického spouštění.

The relation service Dovoluje vytvářet asociace mezi MBeany.

2.3.5 Druhy MBeanů

Managed Bean, neboli MBean je java objekt, který implementuje specifické rozhraní a slouží jako základní kámen technologie JMX. MBean může současně plnit několik funkcí:

- Poskytovatel hodnot atributů
- Poskytovatel hodnot atributů s možností tyto hodnoty měnit
- Spouštěč operací
- Vydavatel notifikací

Každý takto implementovaný a registrovaný MBean může pak poskytovat informace (služby) klientům mimo virtuální stroj. Čili MBeany dokáží poskytovat a nastavovat hodnoty svých atributů a spouštět operace vně virtuálního stroje na požádání z vnějšku virtuálního stroje.

Dle knihy [Sullins(2002)] existuje několik druhů MBeanů, které se liší tím, jak jsou vytvářena jejich JMX rozhraní.

Standard MBean Nejjednodušší druh MBeanu, jehož JMX rozhraní tvoří jeho veřejné metody uvedené v rozhraní, které implementuje.

Dynamic MBean Tento druh MBeanů implementuje specifické rozhraní. Avšak jeho JMX rozhraní lze za běhu programu měnit.

Open MBean Dynamic MBean, který má jednodušší samodokumentovatelnou implementaci.

Model MBean Rovněž druh Dynamic MBeanu, který má za úkol vytvořit generického předka skupině MBeanů.

2.3.6 Notifikace

Oblast Notifikace v technologii JMX je implementací Java událostního modelu. To znamená, že klient implementující specifické rozhraní se může zaregistrovat jako posluchač zpráv (notifikací) z MBeanu (MBean serveru), které registraci umožňuje. Po úspěšné registraci klient dostává zprávy vydávané MBeany.

Při prvních pokusech s notifikacemi a plánovaným rozšířením podporovaných služeb na aplikační server JBoss bylo zjištěno, že aktuálně používaná verze serveru JBoss notifikace nepodporuje. Proto bylo od použití notifikací v řešení sledování a spravování upuštěno.

3 Realizační část

Druhá část této práce je rozdělena do dvou celků:

- Popis implementace na straně serverových služeb
- Popis vytvořeného nástroje pro spravování a sledování služeb

Pro realizaci sledování a spravování služeb v rámci softwarového řešení MPA bylo zapotřebí upravit stávající implementaci služeb a navrhnout nástroj, který bude poskytovat uživatelské rozhraní pro spravování a sledování. Standardní nástroj JConsole se ukázal jako nevyhovující. Z hlediska technologie JMX je služba JMX serverem a nástroj na spravování a sledování JMX klientem.

V první části této kapitoly jsou zmíněny změny provedené ve stávající implementaci služeb. Služby bylo zapotřebí upravit tak, aby mohly registrovat nově vytvořené MBeany. Díky společné inicializaci služeb nebylo zapotřebí zasahovat výrazně do implementací každé služby zvlášť, změna byla provedena na jediném místě.

Konkrétní MBeany, které se registrují, určuje voláním statických metod pomocné třídy typicky jedna řádka zdrojového textu. Zásah do zdrojových kódů služeb je tedy minimální. Složitější implementace je pouze u služeb JLaunch a RMIServer (podrobnosti o této implementaci jsou v kapitole 3.1.1).

V druhé části této kapitoly je popsána implementace nově vytvořeného nástroje s grafickým uživatelským rozhraním. Tento nástroj byl pojmenován ACTMnoitor. Jedná se o program napsaný v jazyku Java. Tento nástroj přehledně zobrazuje stav spuštěných služeb.

Uživatelské rozhraní ACTMonitoru, které je vidět na obrázku 3.1, je tvořeno standardním aplikačním oknem. Toto okno obsahuje strom spuštěných služeb s barevným označením stavu uzlu.

Po kliknutí na uzel MBeanu je v pravé části okna zobrazen editor pro daný uzel. Editor slouží k zobrazení informací o rozhraní MBeanu a dovoluje spouštět metody MBeanu. Pod částí okna určeného pro editory se nachází oblast

pro vypisování zpráv. Za zprávy jsou považovány výsledky volání MBeanů a informace o běhu aplikace ACTMonitor.

3.1 Implementace na straně JMX serveru

Služby, programy operačního systému, jsou napsány převážně v programovacím jazyku Java. JMX server je částí virtuálního stroje a jako takový je částí implementace služeb. Zásahy do implementace služeb bylo možno provádět pouze v Java zdrojových kódech. Díky dobrému, objektově orientovanému modelu je inicializace služeb prováděna na společném místě zdrojového kódu inicializace služeb.

Zároveň s inicializací JMX serveru je veřejné rozhraní tohoto serveru registrováno do služby RMIServer. V RMIServeru má služba obvykle dvě reference. Jednu "normální" referenci, kterou využívají ostatní aplikace k přístupu ke zdrojům poskytovaným službou. A druhou "jmx" referenci, která slouží k přístupu k registrovaným MBeanům služby.

Prozatím není implementována v RMIServeru žádná vazba mezi těmito dvěma referencemi. Toto je poměrně nepříjemná vlastnost RMIServeru, která může vést k problémům, kdy je služba restartována a její "jmx" reference zůstane v RMIServeru, zatímco její "normální" reference je odstraněna. Současné řešení tohoto problému spoléhá na časovou prodlevu při restartu služby. Za tuto dobu by mělo být spuštěno periodické kontrolování referencí v RMIServeru, které neplatnou referenci vyčistí.

Druhým zásahem do implementace služeb bylo přidání registrace nově napsaných MBeanů.

V neposlední řadě bylo zapotřebí implementovat nové MBaeny. Jedná se o MBeany:

- control
- logFile_watcher
- memory_watcher
- runtime

- service_tester
- thread_error_handler

Tyto MBeany poskytují funkcionalitu pro sledování a spravování aplikace, ve které jsou registrovány.

3.1.1 Vytvoření MBeanů

Bylo navrženo a implementováno několik základních MBeanů. Fyzicky byla napsána pro každý MBean třída a její rozhraní se jménem třídy a příponou MBean. Mezi nově vytvořené MBeany jsou rozděleny všechny požadavky na sledování a správu běžící služby.

V tabulce 3.1 je přehled základních MBeanů s jejich poli působnosti.

Název MBeanu	Oblasti působnosti
control	spouštění příkazů
logFile_watcher	správa logování událostí
memory_watcher	správa paměti
runtime	informace o nastavení služby
service_tester	testování, zdali je služba v korektním stavu
thread_error_handler	poskytování informací o spuštěných vláknech

Tabulka 3.1: Přehled základních MBeanů

Control MBean je dynamický MBean a je tvořen na základě instance třídy CommandServer. Tato třída registruje příkazy, které pak lze zadávat do příkazového řádku služby. MBean slouží k vyvolání těchto příkazů bez nutnosti zadávat příkaz do příkazového řádku služby.

LogFile_watcher je standardní MBean, který poskytuje informace o logovaných událostech. Tento MBean dovoluje nastavit logování událostí. Události jsou na různých místech logovány různými instancemi třídy Logger. U instancí třídy Logger lze vzdáleně nastavit množství událostí, které budou zaznamenávat dle jejich priority. MBean umožňuje také uložit aktuální stav vláken do souboru.

Runtime MBean poskytuje informace o spuštěném virtuální stroji a navíc umožňuje zobrazit nastavení MPA aplikace dané souborem .properties.

Service_tester je MBean, pomocí něhož služba testuje stav svého jmx rozhraní. Neúspěšný test se promítá do stavu služby, který je zobrazován v ACTMonitoru.

Thread_error_handler je novější MBean, který slouží k testování, zda-li nedošlo ke stavu známému jako deadlock. Zobrazuje i další informace o vláknech běžící služby.

3.1.2 Zásahy do serverových aplikací

Stávající implementaci služeb bylo zapotřebí upravit tak aby při svém spuštění inicializovaly JMX a poté registrovaly MBeany do MBean serveru virtuálního stroje.

Naštěstí prakticky každá MPA aplikace při spuštění volá speciální inicializaci systému. V rámci této inicializace se děje například:

- Nastavení logování - zaznamenávání událostí do souboru.
- Vyžaduje se přihlášení pomocí jména a hesla.
- Aplikace se registruje v RMIServeru.
- Spuštění vlákna sledující alokovanou paměť programu.
- Spuštění vlákna upozorňující na vznik Deadlocku.
- Inicializace národního prostředí.

Do stávající implementace inicializace aplikace byla přidána i inicializace JMX.

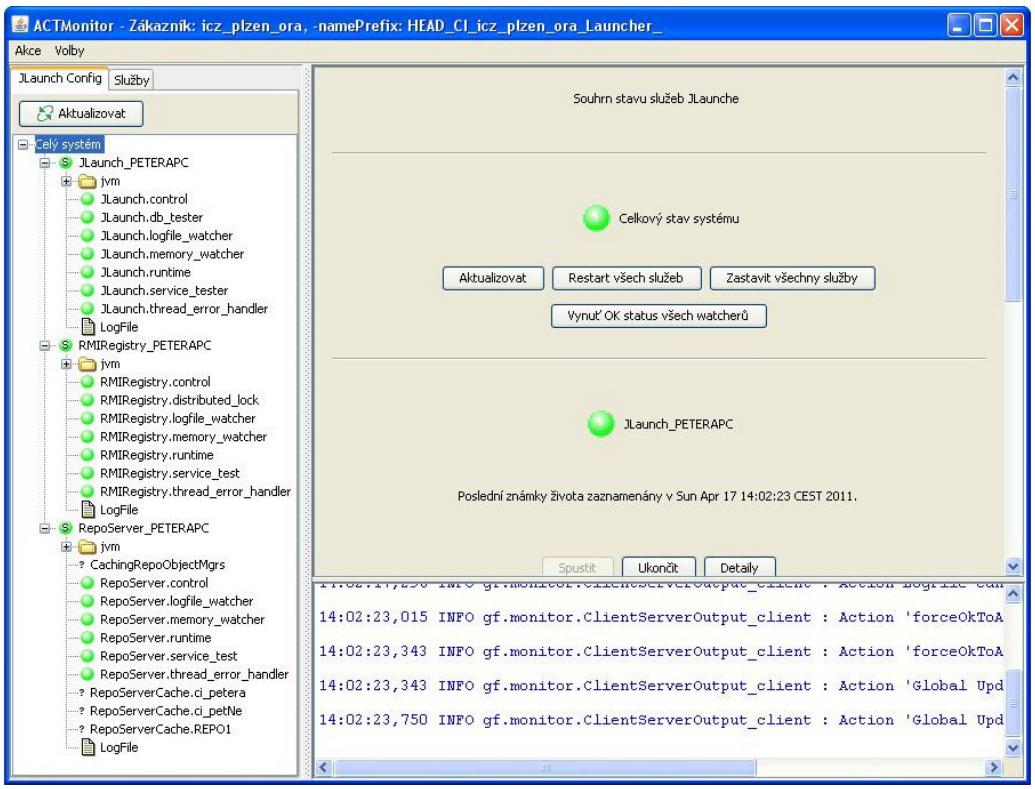
Registrování MBeanů se pak děje po inicializaci systému, a to jednoduchým voláním statických metod třídy, které pomáhá s registrací MBeanů. Zásahy do zdrojových souborů služeb byly tedy minimální. U běžných služeb byly přidány jednotky řádků kódu s registrací MBeanů. Trochu náročnější byla registrace control MBeanu, který je vytvářen z CommandServeru (viz kapitola 3.1).

Složitější, atypickou implementaci vyžaduje registrace MBeanů u služeb RMIServeru a JLaunch. U služby RMIServer lze zaregistrovat MBean server (a tedy i MBeany) až po jejím plném spuštění.

Služba JLaunch je jedinečná tím, že jako jediná služba, může být korektně spuštěna i bez spojení se službou RMIServer. Situace, kdy je služba JLaunch spuštěna a přesto nemá spojení s RMIServem, nastává například při možnosti ztráty spojení s databází a následného restartování všech služeb. Po restartování služby RMIServer je nutné zaregistrovat opět všechny MBeany v podobě MBean serveru služby JLaunch.

3.2 ACTMonitor

Aplikace poskytující přehled běžících aplikací a možnost jejich správy byla pojmenována ACTMonitor. ACT je zkratka technologie Advanced Component Technology, což je technologie společnosti Systema společná pro všechny produkty společnosti vyvíjené spolu s MPA. Monitor pak značí účel aplikace - sledovat běžící služby.



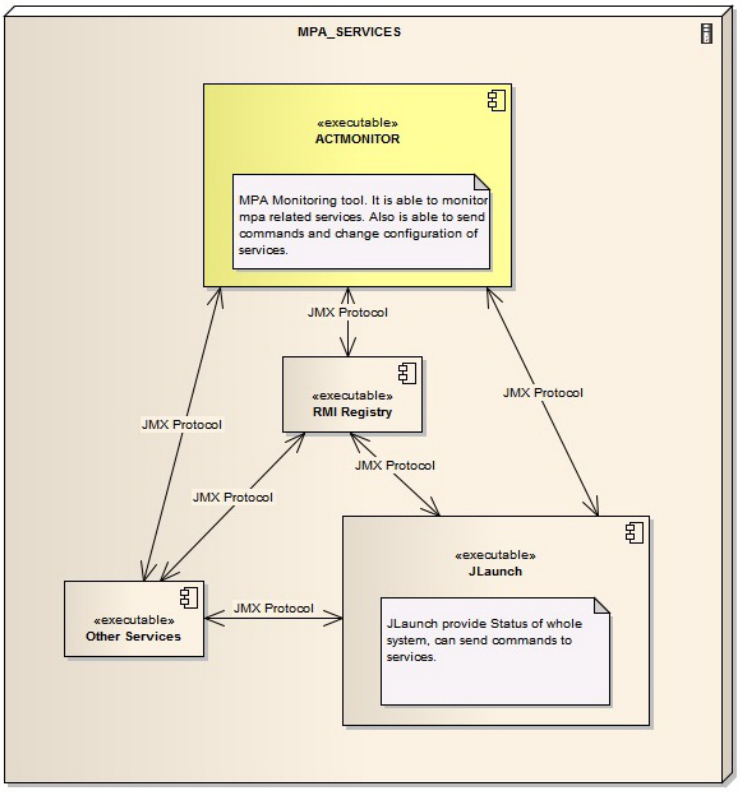
Obrázek 3.1: ACTMonitor

Aplikace je implementována v jazyku Java. Některé konfigurační soubory jsou napsány v jazyce XML. V průběhu implementace byl přidán požadavek na internacionalizovatelnost aplikace. Nyní aplikace podporuje německé, české a anglické jazykové prostředí.

ACTMonitor jako jedna z mála aplikací v softwarovém řešení MPA může být spuštěna a bez větších omezení provozována bez spojení s RMIServerem. Je tomu tak proto, aby bylo možné RMIServer z ACTMnonitoru spouštět a nebylo zapotřebí ACTMonitor restartovat při restartování RMIServeru.

3.2.1 Nasazení aplikace

Aplikace ACTMonitor může běžet na jakémkoliv stroji, který má přístup ke službě RMIServer. Typicky je ACTMonitor spuštěn na stroji se službami a v počítači administrátora systému.



Obrázek 3.2: ACTMonitor - model komunikace

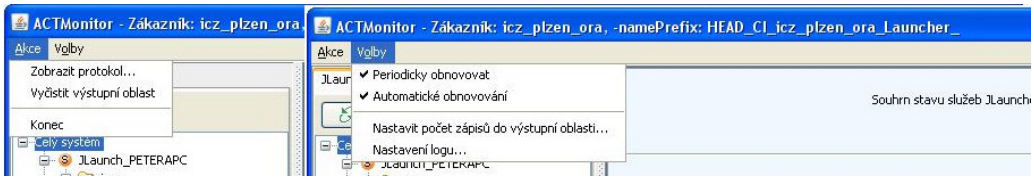
Model komunikace ACTMonitoru se službami je zobrazen na UML diagramu 3.2. ACTMnonitor stejně jako všechny ostatní aplikace v MPA navazuje spojení se službami pomocí RMIServeru. Některé operace nad službami provádí přímo. Jiné operace provází za pomoci služby JLaunch.

Například spouštění a ukončování služeb je prováděno z ACTMonitoru pomocí metody control MBeanu služby JLaunch. Zatím co získávání velikosti aktuálně alokované paměti je prováděno přímo voláním metody MBEanu služby.

3.2.2 Přehled aplikace

ACTMonitor má standardní grafické uživatelské rozhraní. Toto rozhraní je vidět na obrázku 3.1 . Na nejvyšší úrovni je okno aplikace rozděleno na dva panely. V levém panelu je zobrazen strom spuštěných služeb. Pravý panel je rozdělen opět na dva panely. Pravý horní panel slouží k zobrazování podrobností dle volby ve stromu služeb. Pravý dolní panel se nazývá výstupní oblast a jsou sem barevně vypisovány výstupy volání MBeanů služeb a informace o běhu ACTMonitoru.

Menu aplikace ACTMonitor je rozděleno do dvou oblastí: Akce a Volby. Obě položky menu jsou zobrazeny na obrázku 3.3. Položka menu Akce nabízí možnost zobrazit soubor se zaznamenávanými událostmi, smazat zprávy z pravého dolního panelu a ukončit aplikaci.



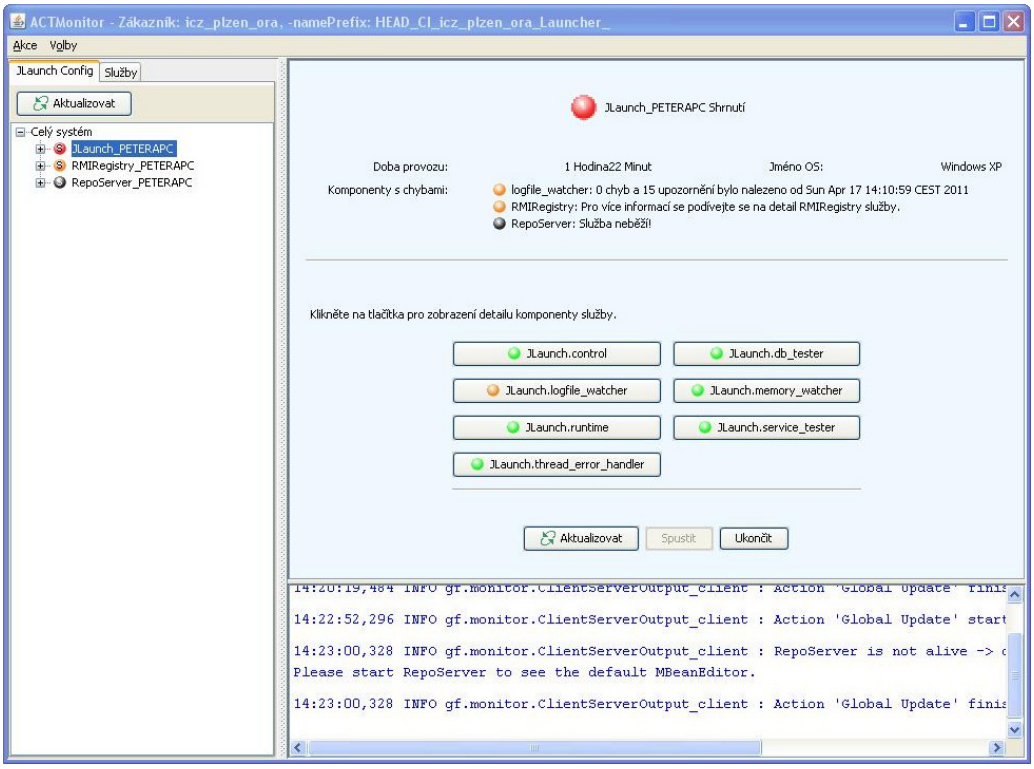
Obrázek 3.3: ACTMonitor - menu

Položka menu Volby nabízí možnost nastavení automatického obnovování aplikace a frekvenci tohoto obnovování. Toto nastavení je důležité, protože sledované služby samovolně nesdělují informace o svém běhu, ale ACTMonitor si musí žádat o jejich stav.

3.2.3 Strom služeb

Levý panel ACTMonitoru je tvořen stromem služeb. Tento strom je dynamicky vytvářen na základě konfigurace služby JLaunch. Pokud služba JLaunch není spuštěna, je zobrazen pouze šedivě kořenový uzel JLaunch.

Velkým přínosem této aplikace je grafické znázornění uzlů služeb a MBeanů. Na obrázku 3.4 jsou zobrazeny různé stavy služeb ve stromu. Pokud aplikace není spuštěna, respektive její reference není v RMIRegistry, je uzel zobrazen šedivě. Pokud služba běží bez znatelných problémů, je uzel služby zobrazen zeleně.

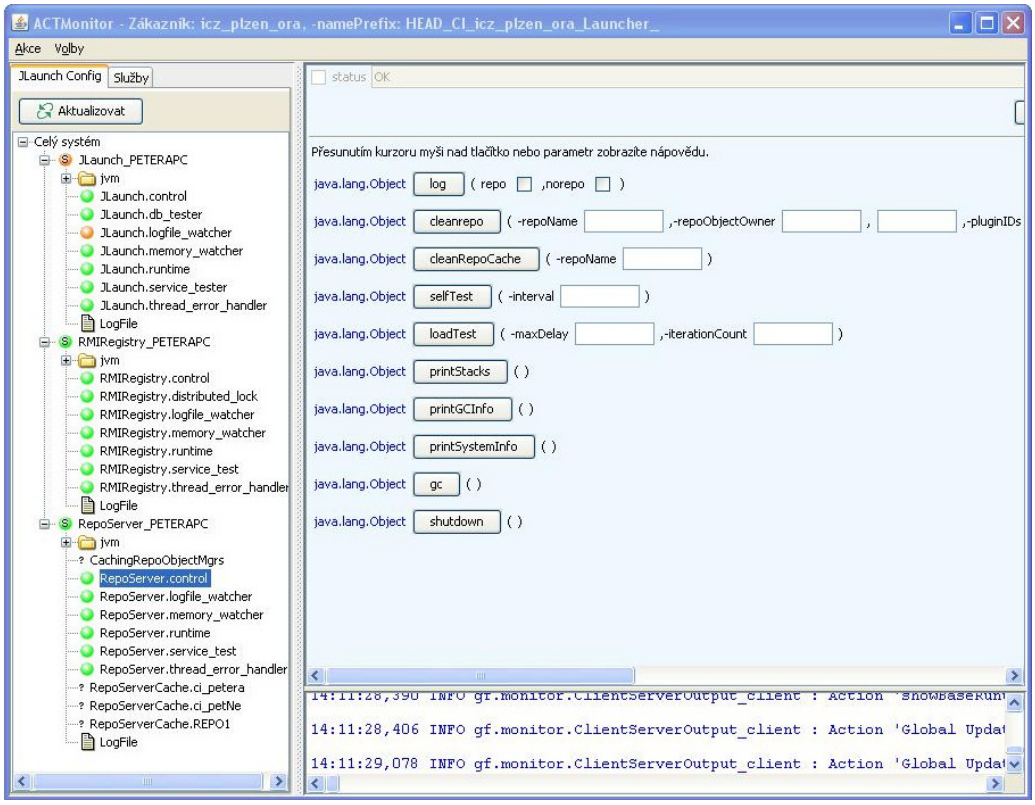


Obrázek 3.4: ACTMonitor - stavy služeb

Existuje několik ukazatelů problémů ve spuštěné službě. Nejčastěji se jedná o logovanou událost na úrovni upozornění nebo chyba. Dalším ukazatelem je například velikost volné paměti. Pokud některý z těchto ukazatelů ukazuje na problémy se službou (dochází volná paměť, nastala a byla logována nečekaná událost), uzel této služby změní barvu na oranžovou nebo červenou. Barva vyšších uzlů je pak dána nejhorší barvou podřízených uzlů. Díky tomu se zjištěný špatný stav služby propaguje až do kořenového uzlu stromu služeb ACTMonitoru.

3.2.4 Editory

Po kliknutí na uzel MBeanu se v pravém horním panelu zobrazí generický editor. Tento editor slouží k zobrazení informací o hodnotách atributů MBeanu, k nastavení hodnot MBeanu a spouštění metod.

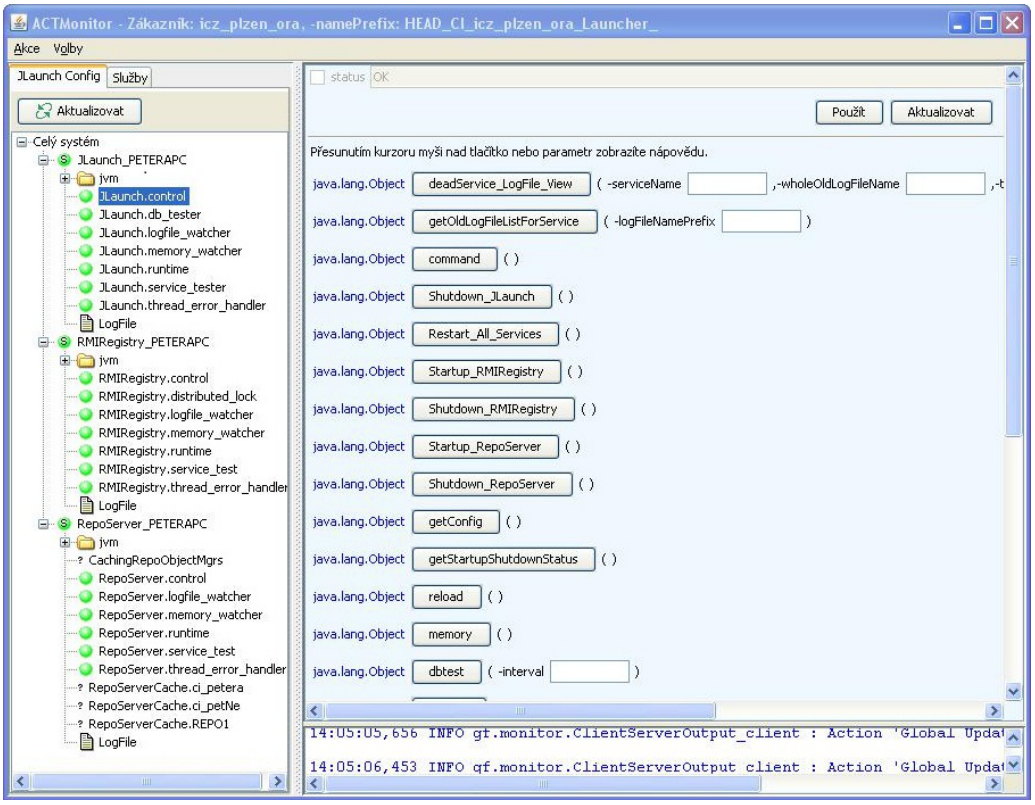


Obrázek 3.5: ACTMonitor - RepoServer control MBean

Tyto editory jsou vytvářeny dynamicky na základě JMX rozhraní daného MBeanu. Dynamické vytváření editorů, je důležitá vlastnost, která v případě změny JMX rozhraní MBeanu předchází potřebě měnit implementaci editoru pro tento MBean, ale stávající mechanismus sám vytvoří, při kliknutí na uzel, editor jiný.

3.2.5 Editory obecných MBeanů

Jak již bylo zmíněno, všechny běžné služby mají registrovány základní, společné MBeany. Tato kapitola popisuje možnosti použití obecných MBeanů prostřednictvím ACTMonitoru. Jsou zde popsány možnosti editoru pro MBeany: control MBean, logFile_watcher MBean, runtime MBean a skupinu standardních MBeanů virtuálního stroje.



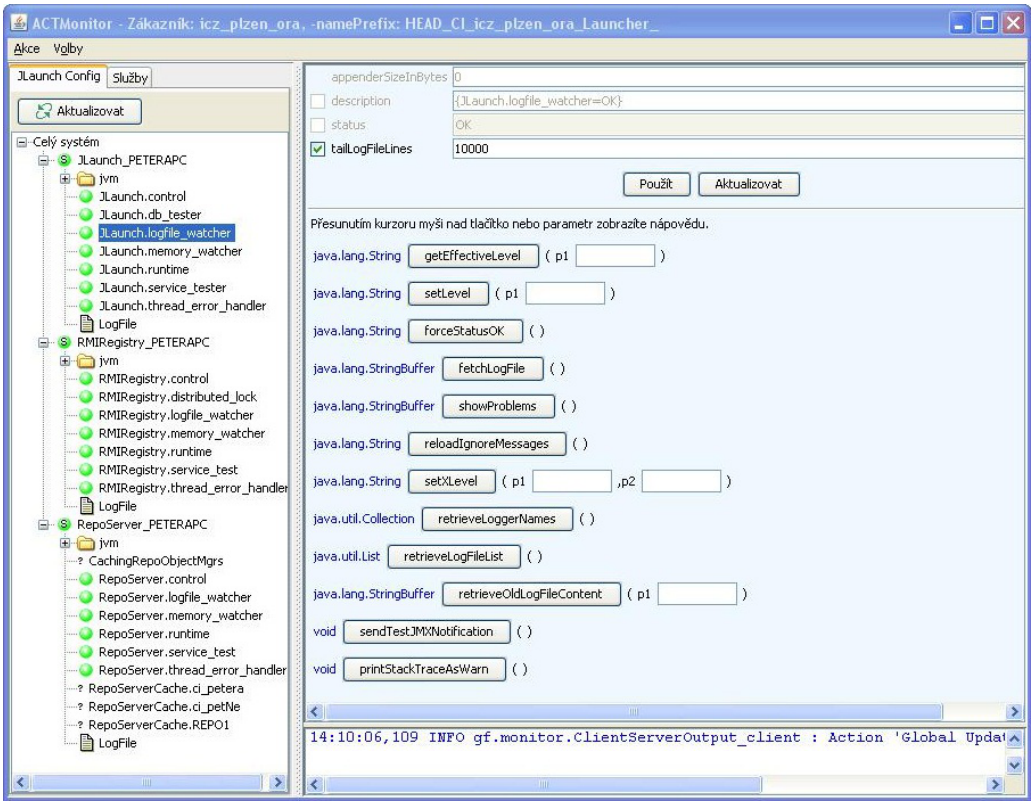
Obrázek 3.6: ACTMonitor - JLaunch control MBean

Na obrázku 3.6 je zobrazen editor control MBeanu služby JLaunch. Tento editor je tvořen dynamicky dle JMX rozhraní MBeanu. Protože control MBean je dynamic MBean (viz kapitola 2.3.5), je teoreticky možné, že se jeho JMX rozhraní může za běhu programu měnit. Prakticky je pouze vytvářen dynamicky při spouštění služby a pak už se nemění. Příkladem takto proměnného JMX rozhraní je rozhraní control MBeanu služby JLaunch. Příkazy Restart a Shutdown jsou přidávány pro každou službu, která je uvedena v konfiguraci služby JLaunch.

Rozhraní control MBeanu je specifické pro každou službu, jak je patrné z obrázků 3.5 a 3.6. I když se jedná o stejnou třídu MBeanu, ale editory jsou odlišné.

Operace, které lze nyní pohodlně spouštět kliknutím v editoru control MBeanu ACTMonitoru, bylo dříve nutné spouštět příkazy v příkazovém řádku služby.

Serverové služby běží typicky v operačním systému Windows Server. Při testování přechodu na novou verzi Windows Server 2008 bylo zjištěno, že v tomto operačním systému je malá podpora klasických "černých" oken příkazového řádku. Vzhledem k tomu, že doposud administrátory Serverových služeb nic nenutilo používat ACTMonitor namísto příkazů v příkazovém řádku, bylo implementováno nastavení, které znemožní zadávání příkazů v příkazovém řádku služby a donutí uživatele použít control MBean v ACTMonitoru.



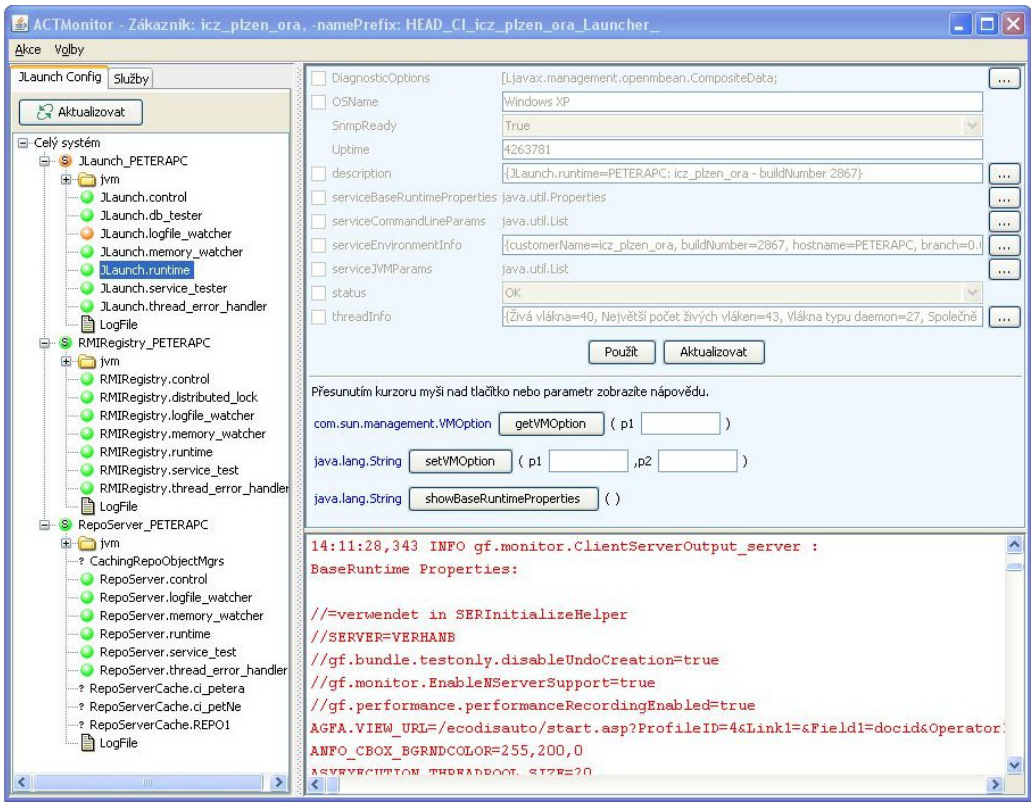
Obrázek 3.7: ACTMonitor - JLaunch logfile MBean

Dalším důležitým editorem je editor pro logFile_watcher MBean. LogFile_watcher je MBean poskytující informace o základním stavu služby na základě zaznamenávaných (logovaných) událostí.

MBean sledující alokovanou operační paměť služby se nazývá memory_watcher. Tento MBean umožňuje nastavit hranici, při které je považován poměr využité paměti oproti alokované paměti za kritický.

Tento MBean byl často využíván při testování indikace stavu služeb. Dovoluje za běhu aplikace měnit kritéria, při kterých je poměr využité paměti

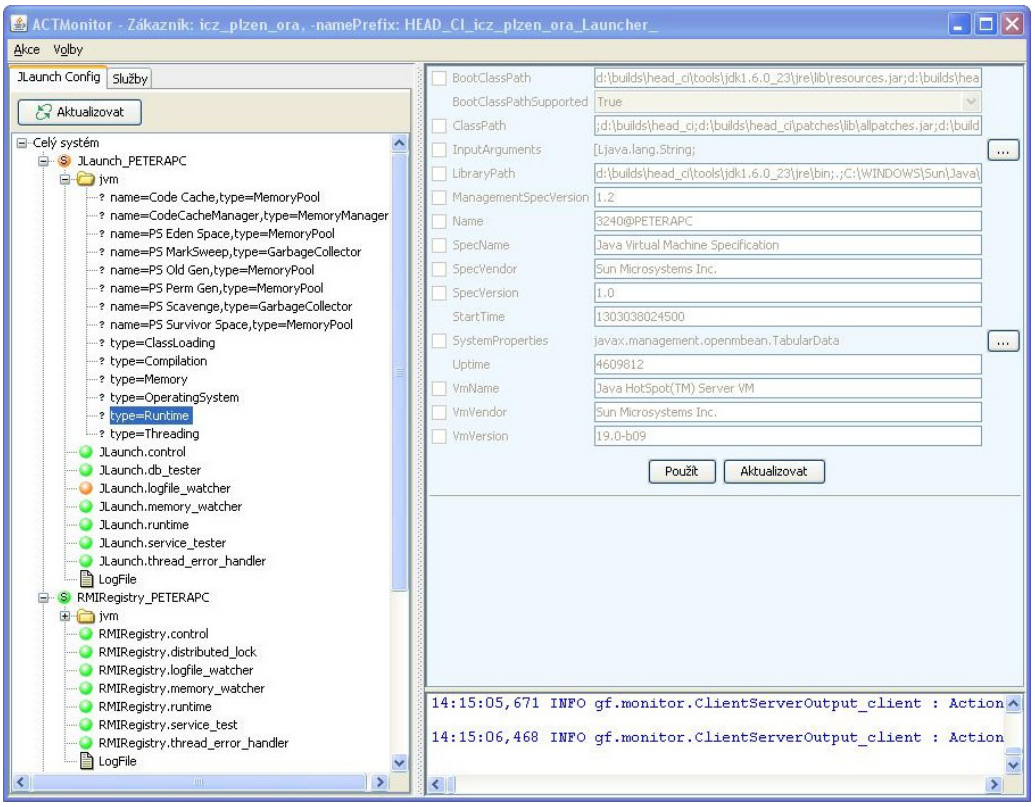
proti alokované paměti považován za kritický. Je-li zapotřebí nastavit stav služby za kritický, a to tak, aby služba sama tento svůj stav indikovala a nebyl ohrožen její běh, stačí nastavit hranici kritického poměru na nízkou hodnotu. Například hodnota, kdy již desetinové využití alokované paměti je kritické, spolehlivě změní optiky stav služby.



Obrázek 3.8: ACTMonitor - runtime MBean

Na obrázku 3.8 je zobrazen stav ACTMonitoru po zavolání metody showBaseRuntimeProperties. Ve výstupní oblasti je vidět červený výstup volání této metody. Metoda showBaseRuntimeProperties je jedna z metod runtime MBeanu. Runtime MBean slouží zejména k získávání informací o běžícím virtuálním stroji, navíc poskytuje i informace o vlastnostech - properties specifických pro aplikace běžící v rámci softwarového řešení MPA.

Ve stromu služeb je také uzel reprezentovaný složkou s názvem jvm. Uzly pod tímto uzlem jsou standardní MBeany virtuálního stroje. Tyto uzly nejsou barevně označeny barevným kolečkem jako ostatní uzly, místo barevného kolečka mají pouze otazník. Tyto uzly se opticky nepodílí na stavu služby. Na obrázku 3.9 je zobrazen editor pro jeden ze standardních MBeanů virtuál-



Obrázek 3.9: ACTMonitor - jvm MBeany

ního stroje, a to Runtime MBeanu. Tento MBean přináší základní informace o spuštěném virtuálním stroji.

Každý uzel služby také obsahuje uzel LogFile. Po kliknutí na tento uzel se zobrazí v novém okně soubor s uloženými zaznamenávanými událostmi. Pro zobrazení obsahu tohoto souboru bylo použito existující komponenty MPA. Byl vytvořen potomek této komponenty, který dovoluje načíst soubor ze vzdáleného umístění.

Ke čtení souboru s událostmi byl použit logFile_watcher MBean, kterému bylo přidáno několik metod.

3.2.6 Implementace aplikace

Aplikace ACTMonitor je napsána v programovacím jazyku JavaSE 6. Vývoj postupně procházel verzemi oprav vývojářského balíku JDK a současně podporovaná verze je Java SE 6 Update 23. K překladu zdrojových kódů byl použit nástroj Maven konkrétně verze 2. K psaní zdrojových kódů byl použit nástroj Eclipse a to postupně od verze 3.3 až po verzi 3.6. K standardnímu sestavení Eclipse byly instalovány některé zásuvné moduly - pluginy.

Pluginy použité při vývoji ACTMonitoru:

FindBugs Potenciální defekty kódu na úrovni přeložených .class souborů hlídá FindBugs plugin. Tento plugin pomáhá odhalit programátorské chyby již při psaní zdrojového kódu v editoru se zapnutým automatickým překladem.

Checkstyle Kontrolu konvencí při psaní zdrojového kódu zabezpečuje Checkstyle plugin. Ten také hlídá některé vlastnosti kódu, jako je například veřejné atributy třídy, velký počet řádků třídy a metody, velký počet argumentů metody a mnoho dalších. Nutí tedy programátora vyhnout se těmto neřestem, které činí zdrojový kód nepřehledný.

Maven Tento plugin podporuje překlad zdrojových kódů pomocí nástroje Maven. Pomáhá zejména při překladu projektu, který používá knihovny třetích stran, které jsou svázány s projektem pomocí programového objektového modelu Mavenu.

QuantumDB QuantumDB plugin je používán k přístupu do databáze prostřednictvím jazyku SQL.

Team Team plugin přináší podporu verzovacího systému PForce.

Aplikace ACTMonitor byla vyvíjena v komerčním prostředí. Proces vývoje se musel řídit pravidly společnosti, pro kterou byla aplikace vyvíjena. Psaní programu bylo rozděleno na úlohy s odhadem trvání obvykle do dvou člověkodní.

Typický proces psaní kódu v rámci jedné této úlohy byl rozdělen do několika kroků. První krok byla vždy analýza zadání. Druhým krokem bylo napsání vlastního zdrojového kódu. Tento zdrojový kód byl poté vložen do verzovacího systému. Do průvodního dokumentu úlohy byla změna popsána a

byly přidány instrukce pro manuální otestování. Dalším krokem byla kontrola zdrojového kódu pomocí takzvaného code-review. Code-review je zjednodušeně řečeno zkontrolování zdrojového kódu osobou, která kód nepsala.

Code-review bylo prováděno zkušenějšími zaměstnanci společnosti. Pokud byly nalezeny defekty kódu, byl zdrojový kód vrácen autorovi na opravu. Po opravě byl proces code-review opakován. Když byl zdrojový kód vyhovující, byl poskytnut zadavateli k softwarovému testování.

3.3 Ověřování kvality implementace

Softwarový produkt Medical Process Assistent je vyvíjen s důrazem na vysokou kvalitu. Produkt je manuálně i automaticky testován. Testování je uzpůsoben celý vývojový proces.

3.3.1 Manualní testování

Programy dodávané v rámci softwarového řešení Medical Process Assistent jsou pravidelně manuálně testovány zaměstnanci oddělení QA (Quality assurance). Každé sestavení produktu je před instalací k zákazníkovi testováno sadou základních manuálních testu a výběrem rozšiřujících manuálních testu. Manuální testy se provádějí dle popisu v dokumentu QS Template.

Formát dokumentu QS Template byl změněn v průběhu implementace. Původní dokument obsahoval seznam kroku, které tester musel provést, aby ověřil chování programu. Nový formát dokumentu obsahuje tabulku o dvou sloupcích. V prvním sloupci je uveden popis akce, kterou musí tester provést. V druhém sloupci je očekávaný výsledek akce (slovní popis ci obrázek). Oba formáty dokumentu obsahují pole pro vyplnění doplňkových údajů o manuálním testu. Do těchto polí se zadává např. verze softwaru, od které je možné tento test provést, motivace k tomuto testu, předpokládaný čas provedení testu. Dokumenty QS Template jsou sdíleny mezi vývojáři a testeři, přičemž právo vytvářet je a editovat mají obě strany. K softwarovému řešení této práce byly napsány desítky dokumentu QS Template.

Po provedení manuálního testu tester zapíše výsledky testu do QS dokumentu. QS dokument je automaticky svázán s dokumentem QS Template.

Pokud byl test neúspěšný, tester navíc vytvoří dokument o chybě s odkazem na QS dokument s neúspěšným manuálním testem.

Manuálně byla také testována každá změna uveřejněná ve verzovacím systému. Toto testování probíhalo na základě ověřování implementace daného úkolu. Každý úkol má svůj průvodní dokument. Tento dokument má odkaz na všechny změny ve verzovacím systém týkající se daného úkolu. Po dokončení úkolu a code review je dokument předán testerovi na otestování s instrukcemi ohledně postupu. Instrukce pro manuální otestování mohou obsahovat jak popis postupu nebo odkaz na dokument QS Template, tak i zmínku, že se jedná o technický úkol, který nemění ani nepřidává funkčnost programu s poznámkou, která oblast softwaru byla změnou postižena. Tester poté provede manuální test řešení úkolu. Pokud test uspěje, tester označí dokument úkolu a změny ve verzovacím systému za otestované.

3.3.2 Automatické testování

Součástí vývoje řešení vzdálené správy aplikací v rámci nemocničního informačního systému byl vývoj, spouštění a správa automatických testu. Pro psaní automatických testu byla využita existující podpora. Z historických důvodu existuje podpora dvou frameworku pro psaní automatických testu. Starší podpora, rozšiřující framework JUint, se nazývá MPAUnit a novější verze, vycházející z frameworku TestNG, ACTUnit.

Podpora MPAUnit vychází ze starší verze JUnit frameworku, kde je zapotřebí dědit od abstraktní testové třídy. MPAUnit má vlastního předka pro testové třídy, který řeší inicializaci prostředí a publikování výsledku testu.

ACTUnit je novější podpora pro psaní automatických testu založená na frameworku TestNG. Testy se píší pomocí anotací. Podpora vznikla v době, kdy psaní testu pomocí anotací nebylo s frameworkem JUnit možné. O tom, že psaní testů pomocí anotací je oblíbené, svědcí fakt, že podpora pro psaní testu pomocí anotací přibyla také do novějších verzí JUnit frameworku.

Většina testu byla napsána s podporou pro testy ACTUnit. Jediná věc, kvůli které byl v některých testech použit MPAUnit frameworkem, je spouštění testu v separátním procesu. Základní chování automatických testu je, že jsou všechny testy spouštěny v jedné instanci virtuálního stroje. MPAUnit dovoluje spustit test v novém procesu, což je zapotřebí ve chvíli, kdy několik paralelních testu využívá globální zdroje. Nebo v případě, kdy test potřebuje

testovat selhální globálního zdroje, který po testu nedokáže obnovit. Případně test potřebuje nastavit specifické podmínky pro svůj běh a navrácení původního stavu virtuálního stroje není triviální.

Automatické testy jsou spouštěny pravidelně každý den. Kvůli poctu podporovaných verzí produktu jsou testy pro danou verzi spouštěny přibližně jednou týdne. Pro každý běh testu je automaticky vytvořen dokument. V případě, že test neuspěl, je manuálně vytvořen dokument o neúspěšném testu s odkazem na dokument o běhu testu. Dokument o neúspěšném testu je naplánován na opravu nebo přímo přiřazen vývojáři na opravu.

Automatické testy monitorovacího nástroje jsou rozděleny do dvou skupin: pomalé a rychlé. Rychlé testy jsou určeny pro spuštění na vývojářském stroji před každou publikací změny kódu. Jejich běh trvá maximálně jednotky minut. Běh pomalých testu zabírá řádově desítky minut, a proto je spouštěn pouze před publikováním rozsáhlejší změny ve zdrojovém kódu.

4 Závěr

Práce vznikla na základě zadání rakouské firmy Systema Human Information Systems GmbH. Praktická část, napsání programu, byla realizována v prostředí firmy Querity s.r.o. v Plzni.

Přínos programu ACTMonitor je nesporný. Správa systému byla do vzniku a realizace této práce náročná a nepřesná. Velice těžko se hledaly v systému příčiny problémů a řešení většinou vedla k restartování služeb či klientských programů. Administrátoři serverových služeb získali v podobě ACTMonitoru mocný nástroj ke sledování a sprovování systému MPA.

Program ACTMonitor je v době odevzdání práce nasazen v produkčním systému. Fáze vývoje je již dokončena a projekt je ve fázi oprav chyb a údržby. Průběžně jsou upravovány funkce vzhledem ke změnám funkcí monitorovaných programů.

Z hlediska bakalářské práce jakožto dokumentu, který může sloužit jako inspirace pro další podobné práce, je třeba alespoň krátce zmínit několik věcí, které se při vývoji aplikace osvědčily.

JMX Použitá technologie JMX je jednoznačně dobrá volba a během vývoje s ní nebyly prakticky žádné problémy.

Úroveň zdrojového kódu Naprosto nejdůležitější při vývoji je psát čitelný, samodokumentovatelný kód. K již napsanému kódu jsem se několikrát vracel a zpočátku se mi stávalo, že jsem se ve vlastním kódu nevyznal. Postupem času jsem si navykl na konvence a zdrojový kód se stal čitelnější. Vzhledem k tomu, že program byl vyvíjen za pomoci verzovacího systému, kdokoliv si mohl mé zdrojové kódy přečíst a musel se v nich také orientovat.

Nástroje pro zjišťování defektů zdrojového kódu Bylo zajímavé zjistit, kolik potenciálních problémů bylo nalezeno ve zdrojovém kódu po zapnutí nástrojů pro zjišťování defektů. Kdyby tyto nástroje byly aktivní už od začátku, jistě bych si ušetřil čas při vývoji.

Test-driven development Test-driven development je způsob psaní zdrojového kódu na základě automatických testů. Programátor nejdříve napíše automatický test a až poté výkonný kód. Zpočátku se zdá psaní

testů zdlouhavé a zbytečné, ale po čase se automatické testy ukazují jako neocenitelná pomůcka při refaktorování a zásahů do stávajícího kódu, který je těmito testy pokryt.

Počítejte s problémy Jistě bych si ušetřil spoustu času upravováním stávající implementace, kdyby byla od začátku navržena tak, že předpokládá, že vše co se může pokazit, tak se pokazí. Například ověřování, zda-li je služba stále dostupná na své adrese, protože mohlo dojít k jejímu násilnému ukončení. Přetížená síť a další možné problémy musí být od začátku brány v potaz.

Na závěr bych rád poděkoval všem kolegům, kteří se podílí na vývoji a údržbě MPA. A v první řadě děkuji inženýru Jiřímu Kimlovi za bedlivé bdění nad projektem ACTMonitor, za hodiny strávené nad code-review mého kódu a za vše, co jsem se od tohoto výborného a zkušeného programátora naučil.

Literatura

[Ajay D. Kshemkalyani(2011)] AJAY D. KSHEMKALYANI, M. S. *Distributed Computing: Principles, Algorithms, and Systems*. New Delhi, India : firstbookstore, 2011. ISBN 0521189845.

[Sullins(2002)] SULLINS, B. *JMX in Action*. Greenwich USA : Manning Publication Co., 2002. ISBN 1930110561.

[Sun(2003)] *JSR 160: JavaTM Management Extensions (JMX) Remote API*. Sun Microsystems, 2003. Dostupné z: `http://jcp.org/en/jsr/detail?id=160`.

[Sun(2009)] *JSR 255: JavaTM Management Extensions (JMXTM) Specification 2.0*. Sun Microsystems, 2009. Dostupné z: `http://jcp.org/en/jsr/detail?id=255`.

[Sun(2002)] *JSR 3: JavaTM Management Extensions (JMXTM) Specification*. Sun Microsystems, 2002. Dostupné z: `http://jcp.org/en/jsr/summary?id=3`.

[Sys(2011)] *Domovská stránka*. Systema Human Information Systems GmbH, 2011. Dostupné z: `http://www.systema.info/en/home/`.

[TRONÍČEK(2011)] TRONÍČEK, Z. *Průvodce architekturou JEE aplikací*, 2011. Dostupné z: `http://www.edumaster.cz/java-developers-solaris-administrators-day/pdf/JSD-2011-GuideToArchitecture.pdf`.