

# VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ  
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY  
DEPARTMENT OF INFORMATION SYSTEMS

## SYSTÉM MONITOROVÁNÍ STAVU PLÁNOVACÍCH ÚLOH

BAKALÁŘSKÁ PRÁCE

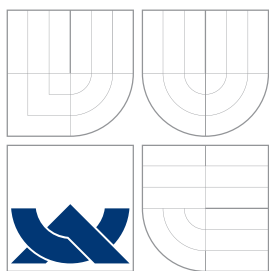
BACHELOR'S THESIS

AUTOR PRÁCE

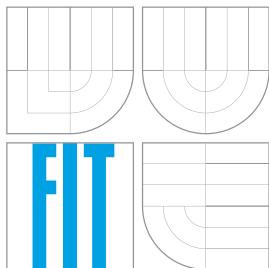
AUTHOR

MARTIN MAGA

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ  
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ  
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY  
DEPARTMENT OF INFORMATION SYSTEMS

# SYSTÉM MONITOROVÁNÍ STAVU PLÁNOVACÍCH ÚLOH

LANNING TASK MONITORING SYSTEM

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE

AUTHOR

MARTIN MAGA

VEDOUCÍ PRÁCE

SUPERVISOR

Ing. ZDĚNEK LETKO, Ph.D.

BRNO 2013

# Abstrakt

.

## Abstract

Výtah (abstrakt) práce v anglickém jazyce.

## Klíčová slova

Java EE 6, Java, SOA, Java Beans, Java Server Faces, Monitorovanie, Twitter, Bootstrap, Systém .

## Keywords

Java EE 6, Java, SOA, Java Beans, Java Server Faces, Monitoring, Twitter, Bootstrap, System.

## Citace

Martin Maga: Systém monitorování stavu plánovacích úloh, bakalářská práce, Brno, FIT VUT v Brně, 2013

# System monitorování stavu plánovacích úloh

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Zdenka Letka

.....  
Martin Maga  
15. dubna 2014

## Poděkování

Veľmi rád by som poďakoval za vedenie mojej bakalárskej práce pánovi Zdenkovi Letkovi a pánovi Martinovi Večeřovi, ktorý mi poskytl rady a podali pomocnú ruku vždy, keď som narazil na problém.

© Martin Maga, 2013.

*Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.*

# Obsah

<b>1</b>	<b>Úvod</b>	<b>3</b>
<b>2</b>	<b>Java Enterprise edition 6</b>	<b>4</b>
2.1	Aplikačný model	4
2.2	Distribovaná viacstupňová aplikácia	5
2.3	Bezpečnosť	6
<b>3</b>	<b>Java EE komponenty</b>	<b>7</b>
3.1	Java EE klienti	7
3.2	The JavaBeans Component Architecture	8
3.3	Webové komponenty	8
3.3.1	Java Persistence API	8
3.3.2	Enterprise Bean	8
3.3.3	Web Services	9
3.4	JavaServer Faces	9
3.5	Enterprise JavaBeans	9
3.6	Web Service	10
3.6.1	"Big" Web Services	10
3.6.2	RESTful Web Service	10
<b>4</b>	<b>JBoss Application Server</b>	<b>11</b>
4.1	História JBoos-u	11
4.2	OptaPlanner	12
4.2.1	NP-úplný problém	13
4.2.2	Výsledky plánovacieho problému	13
4.2.3	Ukážka XML configuračného súboru	15
4.2.4	Optimalizačné algoritmy	16
<b>5</b>	<b>Grafické užívateľské rozhranie</b>	<b>18</b>
5.1	Twitter Bootstrap	18
5.2	Rich Faces	18
5.2.1	Výhody	18
5.3	Rozbor aplikácie	19

5.3.1	Databázová technológia . . . . .	19
5.3.2	Návrh modelu databáze . . . . .	19
5.3.3	Diagram užitia . . . . .	21
5.4	Návrh rozhrania . . . . .	22
5.5	Implementácie . . . . .	22
5.6	Testovanie . . . . .	22
5.7	Vyhodnotenie aplikácie . . . . .	22
<b>6</b>	<b>Záver</b>	<b>23</b>

# Kapitola 1

## Úvod

## Kapitola 2

# Java Enterprise edition 6

V posledných rokoch prevláda tendencia tvorby komplexných informačných systémov, ktoré spracovávajú veľké množstvo dát. Preto sa zvyšuje tak na vývojárov na tvorbu prostriedku, ktoré dokáže takéto systémy ľahko a rýchlo vytvárať. Jedným z takýchto protiedkov je Java Enterprise Edition 6 (Java EE 6). Java EE 6 predstavuje platformu určenú na vývoj webových a podnikových aplikácií.<sup>[4]</sup> Tieto aplikácie sú viacvrstvové z dôvodu lepšej prenositeľnosti, nasaditeľnosti a modifikovateľnosti. Frontend, predstavujúci užívateľské rozhranie a logiku na jeho ovládanie, pozostáva z webových frameworkov, stredná vrstva poskytuje bezpečnosť a transakcie. Najnižšia vrstva poskytuje pripojenie k databázam. Java EE 6 je vhodná pre implementáciu podnikovej logiky, ktorá dokáže byť riadená alebo interagovať s inými podnikovými aplikáciami. Java EE 6 je platformou, ktorá poskytuje širokú škálu aplikačných programových rozhraní (API), ktoré zjednodušujú, zkracujú a znižujú komplexnosť výslednej aplikácie. Jej vývoj neustále napreduje a je spravovaný Java Community process (JCP).

Java EE 6 bola uvoľnená v decembri 2009 a poskytuje ľahké používanie a kompletnú sadu nástrojov na tvorbu podnikových aplikácií. V súčasnosti vyšla ďalšia verzia, ktorá priniesla ďalšie novinky, ktoré môžete dohľadať v online dokumentácii. V ďalších kapitolách si rozoberieme aplikovaný model jazyka, ktoré je veľmi dôležitý pre pochopenie princípu činnosti aplikácií vyvinutých touto platformou. V ďalšej kapitole rozobereme aplikačný model platformy Java EE 6.

### 2.1 Aplikačný model

Java EE je určená pre implementáciu služieb pre zákazníkov, zamestnancov, dodávateľov kohokoľvek, kto prispieva do podniku. Takéto aplikácie sú komplexné a môže byť k nim prístupované z rozličných zdrojov. Pre lepšie zvládanie sú sústredené do stupňov.

Java EE definuje spôsob implementácie služieb, ktoré sú škálovateľné a prístupné.

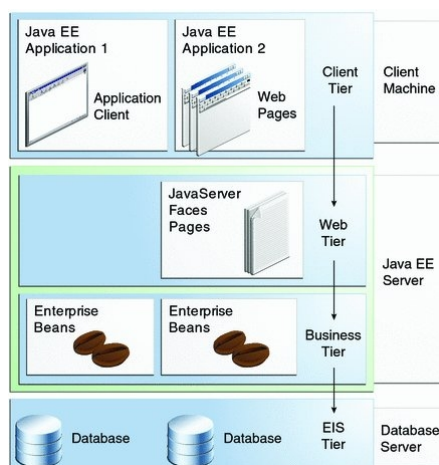


## 2.2 Distribúovaná viacstupňová aplikácia

Java EE používa aplikácie, ktoré sú viacvrstvové(multitier). Aplikačná logika je rozdelená medzi komponenty podľa funkcie.[1] Jednotlivé komponenty sa následne rôzne inštalujú v závislosti, do ktorého stupňa patria. Jednotlivé stupne sa skladajú z rôznych komponent:

- Klientská komponenta, ktorá beží na klientskom počítači
- Webová komponenta, ktorá beží na Java EE serveri
- Podniková komponenta, ktorá beží na Java EE serveri
- Enterprise Information System software bežiaci na EIS serveri

Napriek tomuto rozloženiu býva aplikácia rozložená len do 3 stupňov: Java EE server, klient, databáza. Typicky beží medzi klientskom a databázou častou viac-vláknový Java EE server, ktorý býva označovaný skratkou EIS. Viacstupňovérozloženie môžete názorne vidieť na obrázku č. 2.1. Java EE aplikácia beží na klientskej stanici, býva obvykle re-



Obrázek 2.1: Model Java EE, prevzaté z <http://docs.oracle.com/javaee/6/tutorial/doc/>

prezentovaná tenkým klientom(webovým prehliadačom), nazývaným „thin client“, alebo hrubým klientom, do ktoré je čiastočne vložená logika aplikácia. V strednej časti obrázku sa nachádza Java EE server, na ktorom môžu bežať rôzne technológie v závislosti od požiadavky výslednej aplikácie a možností daného servera. Java EE server môže by reprezentovaný rôznymi dostupnými technológiami, či už sa jedná o open-source riešenia (JBoss, Tomcat, GlassFish) alebo komerčné riešenia (IBM WebSphere, BEA WebLogic), ten obsahuje rôzne komponenty, ktoré so sebou rôzne komunikujú a interagujú na požiadavky klienta a na druhej strane komunikujú s databázovým systémom. Tieto komponenty predstavujú logiku aplikáciu. Posledná časť predstavuje databázový systém, ktorý obsahujú dáta, ktoré klient požaduje pri svojom požiadavku, tento server sa nazýva "EIS".

## 2.3 Bezpečnosť

Java EE vytvára prostriedky, ktoré sú prenositeľné a tak je možné systémy nasadzovať naprieč rôznymi platformami. Poskytuje mechanizmus prístupovej kontroly pravidiel, ktoré sú interpretované, keď je aplikácia nasadzovaná na server. Java implementuje mechanizmus prihlasovania štandardne, čím odpadá programátorovi povinnosť túto časť dodatočne implementovať.

V ďalšej časti sa zameriame na podrobné vysvetlenie jednotlivých Java EE komponent, pretože ich pochopenie bude nevyhnuté pre výslednú aplikáciu.

# Kapitola 3

## Java EE komponenty

Java EE pozostáva z komponent, čo je vlastne sada sebestačného softwaru, ktorý je zhromaždený do Java EE aplikácií a komunikujú s inými komponentami. Java EE definuje nasledujúce komponenty:

- Aplikační klienti a applety bežiaci na klientovi
- Java Servlet, JavaServer Faces, and JavaServer Pages (JSP) komponenty bežia na Java EE serveri[4].
- Enterprise JavaBeans (EJB) komponenty bežia na Java EE serveri[4].

V nasledujúcej časti sa zameriame na vysvetlenie jednotlivých pojmov, pretože sú dôležité z hľadiska pochopiteľnosti výslednej aplikácie.

### 3.1 Java EE klienti

Java EE client je typicky webový klient alebo aplikačný klient.

Typický webový klient pozostáva z dvoch častí:

- Dynamické webové stránky pozostávajúce z rôzneho značkovacieho jazyka (HTML, XML), ktoré sú generované webovými komponentami
- Webovým prehliadačom, ktorý zobrazuje stránky

Typický webový klient sa nazýva "thin" klient, pretože nedotazuje nad databázou, alebo implementuje zložitú business logiku. Všetky tieto činnosti vykonáva Java EE server. Klient len posiela požiadavky Java EE serveru a ten vykonáva dotazovanie a predávanie výsledkov.

Aplikační klienti bežia na klientských počítačoch, kde poskytujú bohatšie užívateľské rozhranie ako webové rozhrania značkových jazykov. Typicky býva vytváraný technológiou Swing<sup>1</sup> alebo Abstract Window Toolkit (AWT)<sup>2</sup>, občas sa vyskytuje aj príkazový riadok ako rozhranie.

---

<sup>1</sup>[http://cs.wikipedia.org/wiki/Swing\\_\(Java\)](http://cs.wikipedia.org/wiki/Swing_(Java))

<sup>2</sup>[http://en.wikipedia.org/wiki/Abstract\\_Window\\_Toolkit](http://en.wikipedia.org/wiki/Abstract_Window_Toolkit)

## 3.2 The JavaBeans Component Architecture

Server a klient tiež zahŕňa komponenty založené na JavaBeans component architecture (JavaBeans components), ktoré správujú toky dát medzi:

- Aplikačným klientom, alebo apletom a komponentami bežiacimi na Java EE serveri
- Serverovými komponentami a databázami

Komunikácia medzi klientom s podnikovým stupňom, ktorý sa nachádza na Java EE serveri, v prípade, že klient beží v prehliadači.

## 3.3 Webové komponenty

Java EE webové komponenty sú súčasťou servletov alebo webových stránok, ktoré sú vytvorené JavaServer Faces technológiu (JSF) and JavaServer pages (JSP) technológiou. Servlety sú javovské triedy, ktoré dynamicky spracovávajú požiadavky a tvoria odpovede. JSP stránky sú textové dokumenty, ktoré pracujú ako servlety ale umožňujú prirodzenejší prístup k vytváraniu statického obsahu. JavaServer Faces technológia stavia na servlety a JSP technológii a poskytuje užívateľské rozhranie komponentný framework pre webové aplikácie.

### 3.3.1 Java Persistence API

Java Persistence API (JPA) je framework jazyku Java, ktorý poskytuje objektovo relačné mapovanie.

### 3.3.2 Enterprise Bean

Enterprise Bean (EB) sú Java EE komponenty, ktoré implementujú Enterprise JavaBeans (EJB) technológiu. Enterprise bean beží v kontajneri EJB. EB je server-side komponentu, ktorá zapuzdruje obchodnej logiky aplikácie. Obchodná logika je kód, ktorý spĺňa účel použitia. Z niekoľkých dôvodov, EB zjednoduší vývoj rozsiahlych, distribuovaných aplikácií. Po prvé, pretože kontajner EJB poskytuje služby na úrovni systému k firme fazuľa, fazuľa vývojár môže sústrediť na riešenie obchodných problémov. Kontajner EJB, je zodpovedný za služby na úrovni systému, ako je riadenie transakcií a autorizácie zabezpečenia.

Po druhé, EB skôr než klientov obsahujú obchodné logiku aplikácie, klient môže developer sústrediť na prezentáciu klienta. Klient developer nemusí kód rutiny, ktoré implementujú obchodných pravidiel alebo prístup k databázam. V dôsledku toho, že klienti sú tenšie, výhoda, ktorá je dôležitá najmä pre klientov, ktorí bežia na malých zariadeniach.

Po tretie, pretože EB sú prenosné komponenty, môže aplikácia assembler budovanie nových aplikácií z existujúcich fazule. Za predpokladu, že používate štandardné rozhranie API, môžu tieto aplikácie bežať na ľubovoľnom kompatibilnom Java EE serveri.

EB sa delia na 2 kategórie:

- Message-driven - Vykoná úloha pre klienta; voliteľne môže implementovať webové služby
- Session - Pôsobí ako poslucháča pre určitý typ správ, ako je API Java Message Service

### 3.3.3 Web Services

Webové služby sú klientske a serverové aplikácie, ktoré komunikujú cez Hypertext Transfer Protocol (HTTP)<sup>3</sup>. Webové služby poskytujú štandardné prostriedky, ktoré sú interoperabilné medzi softvérovými aplikáciami bežiacich na rôznych platformách. Webové služby sa vyznačujú veľkou interoperabilitou a rozšíriteľnosťou, rovnako ako ich strojovým spracovaním, vďaka použitiu XML. Webové služby môžu byť kombinované vo voľne spojených spôsoboch, ako dosiahnuť zložité operácie. Programy poskytujúce jednoduché služby môžu na seba vzájomne a poskytovať sofistikované služby.

## 3.4 JavaServer Faces

JavaServer Faces zodpovedá serverovej strane frameworku pre užívateľské rozhrania vychádzajúce z Javy. JavaServer Faces(JSF) vytvára aplikácie na základe MVC - Model-View Controller<sup>4</sup>. Ďalej treba spomenúť, že pomáha previazať užívateľské údaje so serverom, rovnako ako aj komponenty, ktoré sú znova použiteľné.

Aplikácia, ktorá je vytvára týmto frameworkom pozostáva z webových stránok, grafických komponent, sadou komponent naviazané na serverovú časť. Môže obsahovať rôzne deskriptory a konfiguračné súbory, ktoré nám pomáhajú pri nasadzovaní aplikácie. Základom JavaServer Faces je Facelets, čo je vlastne výzorovo deklaračný jazyk pre JSF. Facelets stránky používajú XHTML 1 a CSS<sup>5</sup>.

## 3.5 Enterprise JavaBeans

Enterprise JavaBeans(EJB) je používaný na vývoj a nasadzovanie komponentne založených aplikácií, ktoré sú škálovateľné a zabezpečené. Typicky obsahuje podnikovú logiku, ktorá pracuje s podnikovou databázou. Informácie o transakčných a bezpečnostných atribútoch býva typicky uložená v metadátach alebo v samostatnom XML súbore. Inštancia sa typicky spravuje za behu kontajnerom. Tento kontajner je prístupný klientovi. JavaBeans poskytuje všetky prostriedky, ktorý súvisia s transakciami stavovým spravovaním, takže ponúkajú možnosť sa sústrediť na podnikovú logiku.

V ďalšej časti sa zameriame na JBoss a OptaPlanner.

---

<sup>3</sup>[http://cs.wikipedia.org/wiki/Hypertext\\_Transfer\\_Protocol](http://cs.wikipedia.org/wiki/Hypertext_Transfer_Protocol)

<sup>4</sup><http://en.wikipedia.org/wiki/Model-view-controller>

<sup>5</sup>[http://en.wikipedia.org/wiki/Cascading\\_Style\\_Sheets](http://en.wikipedia.org/wiki/Cascading_Style_Sheets)

## 3.6 Web Service

Web Service sú klientské a serverové aplikácie, ktoré komunikujú prostredníctvom HTTP protokolu vymeniteľným XML správ. Tieto aplikácie poskytujú interoperabilitu medzi rôznymi platformami naprieč počítačovou sieťou. Web Service umožňuje komunikáciu medzi rôznymi aplikáciami, ktoré bežia na rôznych platformách, napr. Java aplikácie založené na Windowse komunikujú s Net. aplikácia bežiaci na Linuxe. Web services sa delia na 2 kategórie "big"web services a "RESTFul"web services.

### 3.6.1 "Big"Web Services

V Java EE 6 existuje API, ktoré sa nazýva JAX-WS, ktoré umožňuje vytvorenie práve tohto typu web servisu.[4] "Big"web service využíva XML správy, spolu so SOAP a XML jazykom, ktorý definuje architektúru a formát správ. Tento typ Web Service obsahuje definíciu pre Web Service vo formáte WSDL, ktorý je čitateľný aj počítačom. Formát SOAP správ a definíciu jazyka WSDL rozhrania môže znížiť zložitosť vývoja aplikácií, webových služieb.

### 3.6.2 RESTful Web Service

V Java EE 6 existuje pre druhý typ web service API, ktoré sa nazýva JAX - RS. Tento typ web servisu je vhodný pre základné , ad hoc integračné scenáre. REST webové služby, často lepšie integrované s HTTP ako službami založenými na SOAP je , nevyžadujú XML správ alebo definície WSDL služby - API.

## Kapitola 4

# JBoss Application Server

JBoss, čo je vlastne skratka pre JavaBeans Open Source Application Server, v súčasnosti nazývaný WildFly je aplikačný server, ktorý je založený na platforme Java a Java Enterprise Edition.<sup>[3]</sup> Aplikačný server tvorí vrstvu medzi aplikáciami a operačným systémom, pričom rovnako poskytuje aplikácia často využívané funkcie, napr. spracovanie transakcií, výmena správ, atď. . Aplikačné servery podobne ako JBoss Application Server (JBoss AS) je open source. JBoss je Java EE platforma pre vývoj a nasadzovanie Java aplikáciu, webových aplikácií, služieb a portálov.

JBoss AS je napísaný v Jave, preto existuje možnosť ho používať naprieč rôznym platformám. Tento server slúži na vývoj a nasadzovanie podnikových aplikácií, webových aplikácií, služieb a portálov. Tento server je licenovaný pod GNU Lesser General Public License (GNU PL).

### 4.1 História JBoss-u

Všetko naštartoval v roku 1999 Marc Fleury. Pre podporu vývoja middleware InterBohemia sebe rozhodol implementovať jeden zo štandardov J2EE , EJB kontajner. Tým sa zrodil prvý projekt - EJBoss, ktorý sa neskôr premenoval na JBoss. TO niekoľko rokov neskôr sa stal prvým certifikovaným J2EE open source aplikačným serverom. Ďalej by som rád ukázal na vývoj rôznych verzií JBossu:<sup>[9]</sup>

- JBoss AS 4.0 , Java EE aplikačný server 1.4 , je vybavený vloženým Apache Tomcat 5.5 servlet kontajnerom. JBoss môže bežať na mnohých operačných systémoch , vrátane mnohých POSIX platformách (ako GNU/Linux , FreeBSD a Mac OS X) , Microsoft Windows a ďalšie,.
- JBoss AS 5.1 , povolený v roku 2009 , pracuje ako Java EE 5 aplikačný server. Je to menšia aktualizácia hlavnej verzie JBoss AS 5.0, ktorý bol vo vývoji po dobu najmenej troch rokov a bol postavený na vrchole novej JBoss microcontainer.
- JBoss AS 6.0 , bol neoficiálne implementáciou Java EE 6 , vydané 28. decembra 2010 .

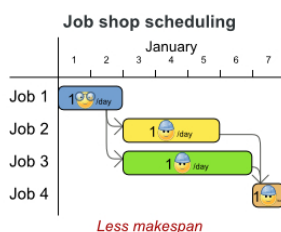
- JBoss AS 7 , bola vydané 12. júla 2011 , len šesť mesiacov po poslednej hlavnej verzii , JBoss AS 6. JBoss AS 7 podporuje rovnakú špecifikáciu Java EE ako posledná verzia, a to Java EE 6. Java EE profil je iba čiastočne implementovaný v JBoss AS 7. Hlavné zmeny viditeľné pre užívateľa sú : oveľa menšiu veľkosť ( menej než polovica z JBoss AS 6 ) a násobné zníženie v čase spustenia.
- JBoss AS 7.1 , aktuálna stabilná verzia bola vydaná vo februári 2012 . Zostávajúce časti EE špecifikácie boli realizované , a táto verzia bola certifikovaná pre EE plnom profile.
- WildFly 8 je priamym pokračovaním na JBoss AS projektu .

V ďalšej časti sa zameriame na OptaPlanner<sup>1</sup>, čo je vlastne systém, pre ktorý je rozhranie navrhované.

## 4.2 OptaPlanner

Optaplanner je framework, kde si môžete vytvoriť pravidlá, ktoré určujú, kedy by mali byť vykonané špecifické akcie. To by mohlo byť vykonané v kóde za použitia podmienok. OptaPlanner je odľahčený open source software a ďalšie pokračovanie frameworku JBoss Drools, ktoré optimalizuje plánovacie problémy.[7] Tieto plánovacie problémy môžu byť nasledujúceho charakteru:

- Plánovanie agendy: plánovanie schôdzok, vymenovanie, práca údržby
- Plánovanie vzdelávania: plánovanie lekcie, kurzov



Obrázek 4.1: Job Shop scheduling, prevzaté z

Obrázok č. 4.1 zobrazuje typické použitie OptaPlanner-u. Môžeme vidieť v nasledujúcom obrázku vystupujú 4 osoby, ktoré vykonávajú nejakú činnosť. Ich činnosť je špecifická a silne závisí od práce predchádzajúcich. Optaplanner sa snaží ich činnosti maximálne optimalizovať a jednotlivé činnosti zvoliť v následnosti tak, aby výsledná práca bola spravená za najkratší možný čas vzhľadom na činnosť, ktorá sa optimalizuje.

<sup>1</sup> <http://www.optaplanner.org/>



### 4.2.1 NP-úplný problém

Každý plánovací problém je NP-úplný problém.[5] NP-úplné problémy sú nedeterministicky polynomiálne problémny, ktoré nie sú riešiteľné v dostupnom čase, pretože sa nepodarilo nájsť deterministický algoritmus. Príkladom NP úloh môžeme považovať: problém obchodné cestujúceho, . . . .

Riešenia poskytnutým týmto frameworkom, ktorý využíva pokročilé optimalizačné algoritmy, sú dosiahnuteľné v reálnom čase. Dosiahnutie v reálnom čase znamená nájdenie 1 alebo viacerých riešení, alebo nenájdenie žiadneho riešenia vzhľadom na poskytnutý čas a optimalizačné algoritmy, ktoré sú implementované.

Každý plánovací problém je definovaný na základe obmedzení, ktoré musia minimálne spĺňať: [2]

- Negatívne "hard"obmedzenie, ktoré nesmie byť porušené
- Negatívne "soft"obmedzenie, ktoré by nemali byť porušené pokiaľ sa dá tomu vyhnúť.

Niektoré problémy môžu obsahovať aj pozitívne podmienky alebo odmeny, ktoré by mali byť splnené pokiaľ je možné ich splniť.

Tieto podmienky definujú skóre plánovacieho problému. Tieto podmienky môžu byť zapísané v Jave alebo v Drools pravidlách, ktoré značne zjednodušujú kód.

Vytvorenie je pomocou pravidiel môže robiť oveľa jednoduchšie spájať mnoho pravidiel s mnohými akciami. Tieto pravidlá bývajú typicky definované pomocou XML súboru.

OptaPlanner pomáha programátorovi riešiť obmedzenie problémov spokojnosti efektívne. Pod kapotou sa kombinuje optimalizačné heuristiky na výpočet skóre.

### 4.2.2 Výsledky plánovacieho problému

Tieto obmedzenia definujú výpočetné skóre problému plánovania. Každé riešenie problému plánovanie môže byť odstupňovaná so skóre.

Plánovanie problému má niekoľko riešení. Existuje niekoľko kategórií riešení:

- Možným riešením je nejaké riešenie, či je alebo nie je ľubovoľný počet obmedzení. Problémy plánovanie mávajú neuveriteľne veľké množstvo možných riešení. Mnoho z týchto riešení sú bezcenné.
- Uskutočniteľným riešením je riešenie, ktoré neporušuje žiadne (negatívne) tvrdé obmedzenia. Niekedy nie sú realizovateľné riešenie. Každý uskutočniteľné riešenie je možné riešenie.
- Optimálnym riešením je riešenie s najvyšším počtom bodov. Problémy plánovanie mávajú jedno alebo niekoľko optimálnych riešení. K dispozícii je vždy aspoň 1 optimálnym riešením, a to aj v prípade , že neexistujú žiadne uskutočniteľné riešenie, a optimálne riešenie nie je možné .

- Najlepším řešením je najít řešení s nejvyšším skóre zistené implementáciou v danom čase.

OptaPlanner podporuje niekoľko optimalizačných algoritmov ako efektívne prehrýzť týmto neuveriteľne veľkým množstvom možných riešení. V závislosti na prípade použitia, niektoré optimalizačné algoritmy dosahujú lepšie výsledky ako ostatné, ale to je nemožné povedať dopredu. Pri plánovaní, je ľahké prepnúť algoritmus optimalizácie, zmenou konfigurácie Solver na niekoľkých riadkoch XML alebo kódu.

### 4.2.3 Ukážka XML configuračného súboru

V nasledujúcom obrázku by som rád ukázal príklad XML configuračného súboru pre OptaPlanner.

Listing 4.1: Test

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <solver>
3   <!--environmentMode>FAST_ASSERT</environmentMode-->
4   <!-- Domain model configuration -->
5   <solutionClass>org.optaplanner.examples.cloudbalancing.domain.
      CloudBalance</solutionClass>
6   <planningEntityClass>org.optaplanner.examples.cloudbalancing.domain.
      CloudProcess</planningEntityClass>
7   <!-- Score configuration -->
8   <scoreDirectorFactory>
9     <scoreDefinitionType>HARD_SOFT</scoreDefinitionType>
10    <simpleScoreCalculatorClass>org.optaplanner.examples.cloudbalancing
        .solver.score.CloudBalancingSimpleScoreCalculator</
        simpleScoreCalculatorClass>
11    <!--<scoreDrl>/org/optaplanner/examples/cloudbalancing/solver/
        cloudBalancingScoreRules.drl</scoreDrl-->
12  </scoreDirectorFactory>
13  <!-- Optimization algorithms configuration -->
14  <termination>
15    <maximumSecondsSpend>120</maximumSecondsSpend>
16  </termination>
17  <constructionHeuristic>
18    <constructionHeuristicType>FIRST_FIT_DECREASING</
        constructionHeuristicType>
19    <!--forager-->
20    <pickEarlyType>FIRST_NON_DETERIORATING_SCORE</pickEarlyType>
21    <!--/forager-->
22  </constructionHeuristic>
23  <localSearch>
24    <acceptor>
25      <entityTabuSize>7</entityTabuSize>
26    </acceptor>
27    <forager>
28      <acceptedCountLimit>1000</acceptedCountLimit>
29    </forager>
30  </localSearch>
31 </solver>
```

Konfigurácie solveru pozostáva z 3 častí:

- Domain model configuration: Musíme Optaplanneru uviesť hlavnú triedu.
- Konfigurácia skóre, ktorá hovorí Optaplanneru ako ma optimalizovať premenné. Pokiaľ používame hard a soft obmedzenia, použijeme "HardSoftScore". Musíme tiež povedať, Planner, ako vypočítať také skóre, v závislosti na našich obchodných požiadavkách. Ďalej sa, sme musíme pozrieť do 2 alternatív pre výpočet skóre: pomocou jednoduchšej implementácie Java, alebo pomocou Drools DRL. Plánovanie bude hľadať riešenie s najvyšším skóre. Budeme používať HardSoftScore, čo znamená, plánovač bude hľadať riešenie s žiadnymi tvrdými obmedzeniami členenie (splňajú požiadavky na hardvér) a pokiaľ možno čo najmenej mäkkých obmedzenia členenie (minimalizovať náklady na údržbu).
- Konfigurácia Optimalizačné algoritmy: Ako by Plánovanie optimalizovať? Nebojte sa o tom teraz: je to dobrý východiskový nastavenie, ktoré funguje na väčšine problémov, plánovanie. To sa už predčí ľudské plánovača a väčšina in-house implementáciou. Použitie Planner porovnávací toolkit, môžete vyladiť, aby to bolo ešte lepšie výsledky.

#### 4.2.4 Optimalizačné algoritmy

V nasledujúcej časti textu si ukážeme optimalizačné algoritmy, ktoré používa optaplanner.

- First FIT - To je veľmi jednoduchý greedy algoritmus aproximácie. Algoritmus spracováva položky v ľubovoľnom poradí. Pre každú položku, pokúsi sa umiestniť na položku v prvom koši, ktorý sa môže ubytovať položku. Ak nie je nájdený žiadny bin, otvára novú priehradku a kladie položku v rámci nového zásobníka.
- First FIT Decreasing
- First FIT Decreasing + heuristic local search
  - Hill Climbing - hill climbing je matematická optimalizácia technika, ktorá patrí do rodiny miestneho vyhľadávania. Jedná sa o iteratívny algoritmus, ktorý začína s ľubovoľným riešením problému, potom sa pokúsi nájsť lepšie riešenie tým, že postupne mení jeden prvok riešenia. Ak zmena vytvára lepšie riešenie, postupné zmeny je, aby nové riešenie, opakovať až do žiadne ďalšie zlepšenie možno nájsť. [6]
  - Tabu Search - Tabu search používa miestnej alebo susedstve vyhľadávania postup iteratívne presunúť z jedného možného riešenia x k lepšiemu riešeniu x v susedstve x, kým sa niektorá zastavenie kritériom bol splnený (Všeobecne platí, že obmedzenia pokus alebo prah skóre). Miestne postupy vyhľadávania sa často stávajú uviaznutými v zlej bodovania oblastiach alebo v oblastiach, kde skóre plošine. V snahe vyhnúť sa týmto nástrahám a preskúmať oblasti

hľadaného miesta , ktorá by mala zostať bez prehliadky inými miestnymi postupmi vyhľadávanie , tabu search starostlivo skúma okolí každého roztoku ako hľadanie postupuje . Riešenie prijatí do novej štvrti ,  $N^*(x)$  , sú určené pomocou pamäťových štruktúr . Pomocou týchto pamäťových štruktúr .

- Simulated Annealing - Simulované žíhanie je všeobecný algoritmus pre globálne optimalizačné problémy lokalizovať dobré priblíženie k globálnej optimálnej danej funkcii vo veľkom vyhľadávacieho priestoru. To sa často používa pri hľadaní priestorov je diskkrétne ( napr. všetky výlety , ktoré navštevujú danú množinu miest ) . U niektorých problémov , môže simulované žíhanie byť účinnejšie ako vyčerpávajúci zoznam - za predpokladu , že cieľom je iba nájsť prijateľne dobré riešenie v stanovenú dobu , skôr ako tým najlepším možným riešením .

V ďalšej kapitole by som rád uviedol problematiku užívateľského rozhrania.

## Kapitola 5

# Grafické užívateľské rozhranie

V tejto kapitole sa zameráme na problematiku užívateľského rozhrania, ktoré vlastne je reprezentované Web services. Toto rozhranie bude umožňovať nahrávať pravidlá, zobrazovať výsledky, spúšťať, pozastávať a zobrazovať detaily úloh. Keďže táto výsledná aplikácia by mala byť použiteľná aj na mobilnom telefóne bol vybratý štýlovací framework Twitter Bootstrap, ktorý značne uľahčuje tvorbu takéhoto rozhrania.

### 5.1 Twitter Bootstrap

Twitter Bootstrap je veľmi jednoduchý a voľne dostupný súbor nástrojov pre vytváranie moderného webu a webových aplikácií.[8] Ponúka podporu najrôznejších webových technológií HTML , CSS , JavaScript a mnoho prvkov , ktoré je možné ľahko implementovať do svojej stránky. Pre použitie Twitter Bootstrap sú nutné základné znalosti HTML a CSS. Interaktívne prvky ako sú tlačidlá, boxy , menu a ďalšie kompletne nastavené a graficky spracované elementy je možné vložiť iba pomocou HTML a CSS .

### 5.2 Rich Faces

Rich faces predstavuje open-source Ajax knižnicu, ktorá predstavuje rozšírenie pre Java-Server Faces. Umožňuje integráciu schopností ajaxu do enterprise aplikácií.

#### 5.2.1 Výhody

Výhodou tohto súboru nástrojov je jednoduché spracovanie akéhokoľvek používateľského rozhrania vo webovej aplikácii a nerozhoduje , či to je napríklad používateľské rozhranie v administrácii back-endových alebo front-endových aplikácií.

Podrobné vysvetlenie jednotlivých komponent nájdete na nasledujúcej adrese <http://getbootstrap.com/>, rovnako aj s príkladmi použitia. V nasledujúcej časti prejdeme na samotný návrh užívateľského rozhrania.

## 5.3 Rozbor aplikácie

Výsledná aplikácia bude zobrazovať priebežné výsledky výpočtu frameworku optaplaner. Tento framework bude pre túto prácu optimalizovaný pre úlohu N Dám, ktorú bude schopný riešiť. Bude sa deliť na aplikáciu, ktorá predstavuje užívateľské rozhranie vytvorené prostredníctvom technológie Java Server Faces v kombinácii s Rich Faces nastýlované prostredníctvom frameworku Twitter Bootstrap, ktoré zároveň zabezpečuje prenositeľnosť rozhrania na mobilné telefóny. Užívateľské umožňuje zobrazovanie a spravovanie úloh, organizácií, do ktorých náležia jednotliví užívatelia a užívateľov. Každá časť systému bude sprístupnená podľa príslušnosti užívateľa k užívateľskej role. Z tohto rozhrania bude môcť užívateľ pokiaľ mu to užívateľská rola dovoľuje pridať úlohu, ktorú môžeme následne spustiť alebo pozastaviť. Spustenie prebehie zavolaním služby web service, ktorá obsahuje potrebné prostriedky na spustenie úlohy. Web service následne zavolá enterprise bean, v ktorej prebieha spracovanie danej úlohy. Výsledok výpočtu sa priebežne ukladá do databázy. Výsledku sa následne priebežne zobrazuje v užívateľskom rozhraní. Užívatelia majú prístup k obmedzenému počtu úloh, zároveň môžu vykonávať obmedzené akcie a to nasledovne:

- Administrátor - má prístup ku všetkým úlohám v systéme, úlohy môže editovať, vytvárať, mazať, publikovať a odpublikovať, môže vytvárať, mazať a editovať užívateľov, rovnaké možnosti má aj s organizáciami
- Plánovač - má prístup k úlohám v rámci svojej organizácie, môže vytvárať, editovať, mazať úlohy, publikovať a odpublikovať
- Čitateľ - úlohy môže len zobrať v rámci svojej organizácie, publikovať, odpublikovať

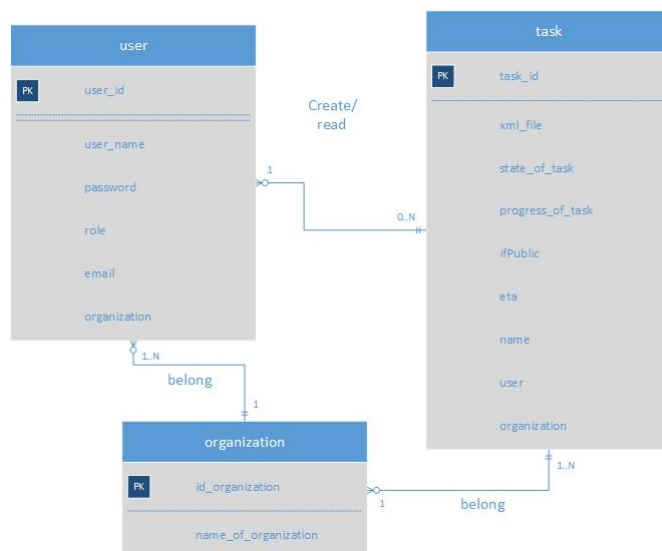
Výsledná aplikácia bude nasadená na aplikačný server JBoss.

### 5.3.1 Databázová technológia

Aplikácia potrebuje pre spracovanie úloh, organizácií a užívateľov databázu. Pre potreby bakalárskej práce bol vybratý relačný open-source databázový model MySQL. MySQL databázová technológia je veľmi vhodná pre malé a stredne veľké aplikácie, čo tá naša je, rovnako poskytuje dobrý výkon pri vykonávaní transakcií, umožňuje vytvárať procedúry, databázové triggere a jej inštalácia je pomerne jednoduchá a nezaberá veľa diskového priestoru, rovnako je MySQL multiplatformová, keďže je možné ju nasadiť na systémy s operačným systémom windows, linux, max os. Medzi nevýhody tejto technológie patrí neefektívna práca s databázovými transakciami, neefektívne ukladanie veľkého množstva dát.

### 5.3.2 Návrh modelu databázy

Na nasledujúcom obrázku je ukázaný ER diagram, ktorý bol použitý pre databázu:



Obrázek 5.1: ER diagram

Tento obrázok zobrazuje jednotlivé entity, ktoré sú potrebné na uloženie v databáze, každá z nich má určité položky. ER diagrama sa skladá z 3 entít: user - entita, ktorá reprezentuje užívateľ, task - entita, ktorá reprezentuje úlohu a organization - entita, ktorá reprezentuje organizáciu. Výsledný návrh odpovedá skutočnosti, že každý užívateľ musí byť súčasťou organizácie, rovnako môže mať vytvorené 0 až N úloh. Taktiež pre zjednodušenie je každá úloha priradená priamo organizácií pre zlepšenie rýchlosti získania výsledkov a zjednodušenia ich nájdenia. Každá entita obsahuje primárny kľúč (jedná sa o silné entitné množiny), ktorý je odvodený od názvu a začína predponou „id\_“ a pokračuje názvom entity. Poďme sa pozrieť bližšie na jednotlivé entity. Entitná množina organization obsahuje 2 položky jednou z nich je primárny kľúč a ďalšou názov organizácie podľa, ktorej sú zaraďovaný jednotliví užívatelia. Ďalej prejdime k entitnej množine user. Táto entita má rovnako primárny kľúč. Ďalej obsahuje položku pre užívateľské meno (user\_name), heslo (password), email, užívateľskú rolu (role) a cudzí kľúč organization, ktorý obsahuje na organizáciu. Nakoniec prejdime k entitnej množine task. Táto entitná množina obsahuje primárny kľúč, ďalej obsahuje xml súbor, ktorý reprezentuje danú úlohu (v našom prípade N dām), stav úloh (state\_of\_task, ktorý reprezentuje rôzne stavy úlohy), ktorý si podrobnejšie rozobereme. Úloha sa môže nachádzať v jednom z nasledujúcich stavov:

- NEW - úloha bola vytvorená
- MODIFIED - xml súbor bol modifikovaný
- WAITING - úloha čaká na spracovanie
- IN\_PROGRESS - práve prebieha výpočet

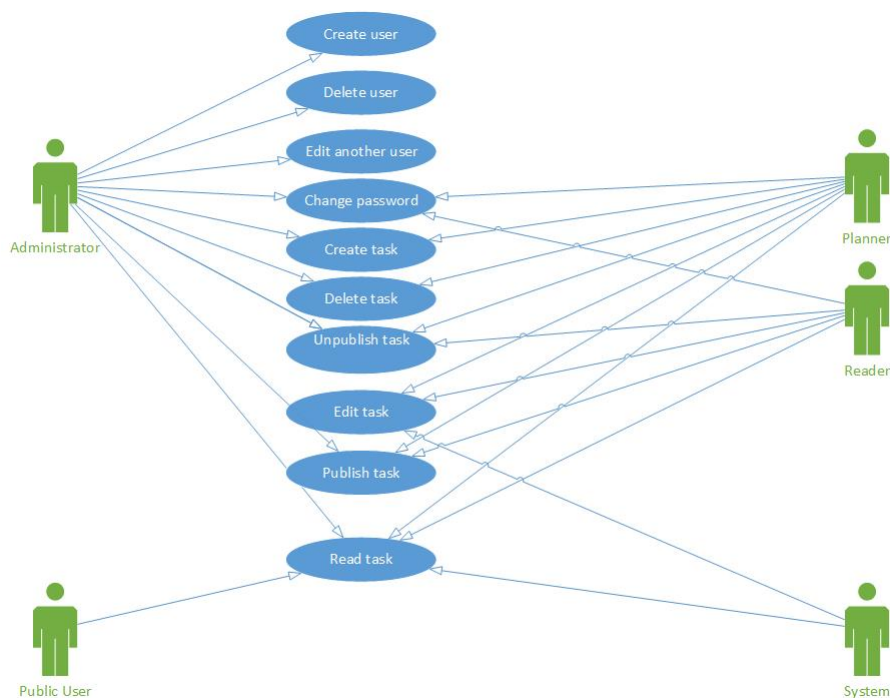


- PAUSED - úloha je pozastavená
- COMPLETE - úloha je dokončená

Entitná množina task ďalej obsahuje položku, ktorá percentuálne hodnotí stav výpočtu úlohy(`progress_of_task`), čas do skončenia výpočtu úlohy(`eta`), nastavenie úlohy na súkromnú alebo verejnú(`ifPublic`), názov úlohy(`name`) a cudzie kľúče `user`, ktorý odkazuje na užívateľa, ktorým bola úloha vytvorená a `organization`, ktorá odkazuje na organizáciu užívateľa, ktorým bola vytvorená. V ďalšej kapitole sa pozrieme na use case diagram.

### 5.3.3 Diagram užívania

Nasledujúci obrázok ukazuje príklady užívania systému: Na nasledujúcom obrázku č.5.2



Obrázek 5.2: UseCase diagram

môžeme identifikovať 4 užívateľov systému. Public user je verejný užívateľ, ktorý môže čítať úlohy, ktoré sú nastavené ako verejné(public). Systém môže načítať úlohy z databázy pre potreby výpočtu, z ktorými následne pracuje, a potom výsledky ukladá do databázy, teda edituje úlohy(tasky). Čitateľ(Reader) môže úlohy čítať, publikovať a odpublikovať. Plánovač môže úlohy čítať, vytvárať v databáze, mazať, editovať už vytvorené úlohy v rámci svojej organizácie, publikovať a odpublikovať úlohy a meniť vlastné heslo. Administrátor môže vytvárať úlohy, mazať úlohy, editovať úlohy, publikovať a odpublikovať úlohy. Rovnako si môže meniť heslo, vytvárať, mazať editovať organizácie a užívateľov.

## **5.4 Návrh rozhrania**

Výsledné rozhranie kladie dôraz na jednoduchosť a prehľadnosť zobrazených úloh. Z tohto dôvodu boli implementované mechanizmy vyhľadávania úloh, organizácií a užívateľov. Rovnako možnosti lexikografického triedenia. Po prihlásení do systému Jednotlivé možnosti sú následne zakomponované do záložiek, v ktorých je prístupná príslušná funkčnosť. Výsledné rozhranie je prenositeľné aj na mobilné zariadenie, čo je spôsobené použitím frameworku Twitter Bootstrap.

## **5.5 Implementácie**

## **5.6 Testovanie**

## **5.7 Vyhodnotenie aplikácie**

## Kapitola 6

## Záver

# Literatura

- [1] Arun Gupta: *Java EE 6 Pocket Guide*. O'REILLY, 2012, ISBN 978-1-449-33668-4.
- [2] Bali, M.: *Drools JBoss Rules 5.X Developer's Guide*. Packt Publishing, 2013, ISBN 978-1782161264.
- [3] Davis, T. M. S.: *JBoss at Work: A Practical Guide*. O'Reilly Media, 2006, ISBN 978-596-00734-8.
- [4] Jendrock, E.; Cervera-Navarro, R.; Evans, I.; aj.: The Java EE 6 Tutorial [online]. <http://docs.oracle.com/javaee/6/tutorial/doc/>, 2013-11-03 [cit. 2013-10-25].
- [5] Johnson, M. R. G. D. S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. Macmillan Higher Education, 1979, ISBN 978-0716710455.
- [6] Michiels, J. K. E. A. W.: *Theoretical Aspects of Local Search*. Springer, 2007, ISBN 978-3540358534.
- [7] Thilo, M. O. J. T. C. R. J.: Twitter Bootstrap [online]. <http://getbootstrap.com/>, 2013-12-16 [cit. 2013-12-27].
- [8] Thilo, M. O. J. T. C. R. J.: Twitter Bootstrap [online]. <http://getbootstrap.com/>, 2013-12-16 [cit. 2013-12-27].
- [9] WWW stránky: JBoss. <https://docs.jboss.org/author/display/AS71/Documentation>, 2013-12-16 [cit. 2013-12-27].