MASARYKOVA UNIVERZITA

FAKULTA INFORMATIKY

Užívateľské rozhranie pre EGEE službu Logging & Bookkeeping

Diplomová práca

Bc. Tomáš Repel

Brno 2010

Prehlásenie

Prehlasujem, že táto práca je mojím pôvodným autorským dielom, ktoré som vypracoval samostatne. Všetky zdroje, pramene a literatúru, ktoré som pri vypracovaní používal alebo z nich čerpal, v práci riadne citujem s uvedením úplného odkazu na príslušný zdroj.

Ďakujem vedúcemu mojej diplomovej práce, Mgr. Miroslavovi Rudovi, za cenné rady, návrhy a pripomienky. Moja vďaka tiež patrí Mgr. Alešovi Křenkovi PhD., ktorý sa aktívne zapájal do konzultácií spojených s touto prácou a prispel nejednou inšpiratívnou myšlienkou.

Zhrnutie

Táto práca si kladie za cieľ vytvoriť užívateľsky prívetivé grafické rozhranie pre Logging & Bookkeeping službu. Toto rozhranie má za úlohu nielen získať a zobraziť dáta zhromaždené touto službou, ale aj poskytnúť možnosť off-line prezerania dát a implementovať funkcionalitu pre manipuláciu s týmito dátami.

Kľúčové slová

EGEE, grid, Logging & Bookkeeping, grafické užívateľské rozhranie, Java

Obsah

1 Úvod 1
2 Gridové služby 3
2.1 Vznik a vývoj gridových služieb 3
2.2 Definícia gridu 4
2.3 Kategorizácia gridov 4
2.4 Virtuálne organizácie 5
3 Enabling Grids for E-SciencE 6
3.1 História projektu EGEE 6
3.1.1Združenie CESNET 6
3.1.2Projekt EGEE 6
3.2 Middleware gLite 7
3.3 Monitorovanie úloh v gride založenom na gLite 8
4 Logging and Bookkeeping služba 10
4.1 Jobid a jeho funkcia 10
4.2 Zhromažďovanie udalostí 11
4.2.1Local logger a inter logger 12
4.2.2L&B Proxy 12
4.3 Formát udalostí 13
4.4 Radenie udalostí 13
4.5 L&B server 13
4.6 Bezpečnosť a kontrola prístupu 14
4.7 Aktuálny stav 14
5 Užívateľský prístup k L&B službe 15
5.1 Metódy klientského API používajúceho webové služby 16
5.2 Webové služby 17
6 Grafické užívateľské rozhranie 18
6.1 Použitie GUI 20
6.1.1Ukladanie úloh pre prácu off-line 21
6.1.2Rozdelenie dotazu podľa časového razítka 22
6.2 Koncept virtuálnych zložiek 23
6.2.1Mazanie úloh prostredníctvom virtuálnych zložiek 24
6.2.2Spracovanie dotazu virtuálnej zložky 25
6.2.3Vytvorenie SQL podmienky pre virtuálnu zložku 25
6.3 Vstupná konfigurácia 26
6.4 Trieda JobDetail 27

6.5 Uloženie stavu úlohy do lokálnej databáze 28
6.5.1 Spracovanie viachodnotových parametrov 29
6.5.2 Uloženie špeciálnych parametrov 29
6.6 Knižnica obecných tried 30
6.6.1 Trieda WsdlParser 30
6.6.2 Trieda QueryCondition 30
6.7 Databázová schéma 31
6.8 Ošetrovanie duplicitných záznamov 32
6.9 Štatistické údaje o prijatých úlohách 33
6.9.1 Špecifikácia potrebných údajov 34
6.9.2 Rozhranie modulu pre štatistiky 35
6.9.3 Príklady štatistických modulov 37
6.9.4 Implementácia modulu ExemplaryStatsPlugin 38
6.10 Zobrazenie detailu o úlohe 39
6.10.1 Moduly pre zobrazenie detailu úlohy 40
6.10.2 Implementácia pluginu DefaultDetailPlugin 41
6.11 Indexovanie dát v databáze 42
6.11.1 Využitie indexov pri urýchlení databázových operácií 43
6.12 Konzument webovej služby L&B servera 44
6.12.1 Rozhranie pre konzumenta L&B služby 45
6.12.2 Metóda saveResult 46
6.12.3 Existujúce implementácie konzumenta 47
7 Záver 49
Literatúra 50
Príloha A 52
Príloha B 53
Príloha C 54
Príloha D 55

Kapitola 1

Úvod

Vedecko-technický rozvoj prináša nové možnosti uplatnenia počítačov takmer vo všetkých oblastiach ľudského snaženia. Počítače sa využívajú v čoraz zložitejších experimentoch, získavajú sa väčšie objemy dát, ktoré je nutné, opäť s pomocou počítačov, analyzovať a uložiť. Nové znalosti zvyčajne vedú k ďalším experimentom, kruh stále sa zvyšujúcich nárokov na výpočetnú silu sa uzatvára. Vytvárajú sa výpočetne náročné modely čoraz presnejšie simulujúce komplexné vedecké a technické problémy. Obdobné tvrdenia platia nielen pre vedeckú sféru. Náročné algoritmy a veľké objemy dát sa využívajú napríklad pri analýze statiky veľkých stavieb a mostov, pri výpočte aerodynamiky lietadiel a automobilov alebo na bežnú predpoveď počasia. Pokrok v kvalite a rýchlosti procesorov a ďalších súvisiacich komponent iba zmierňuje zvyšujúci sa dopyt po voľných výpočetných zdrojoch.

Jedným z riešení nedostatku výpočetných kapacít je spojenie existujúcich zdrojov prostredníctvom sieťovej infraštruktúry tak, aby bol dosiahnutý požadovaný výkon. Zdroje pritom môžu patriť rôznym inštitúciám v geograficky vzdialených oblastiach. Komplex takto spolupracujúcich počítačov sa nazýva grid. Základným softvérovým kameňom gridových služieb je middleware. Je to akási stredná vrstva, ktorú musí mať nainštalovanú každá logická komponenta gridu. Táto služba primárne pôsobí ako zjednocujúci prvok medzi rôznymi hardvérovými a softvérovými uzlami, umožňuje štandardizovaný prístup a zabezpečuje kooperáciu a koordináciu jednotlivých komponent gridu.

Najvýznamnejší európsky vývoj v oblasti gridových služieb sa sústreďuje okolo projektu EGEE[1]. V rámci tohto projektu bol vyvinutý middleware s názvom gLite[2], ktorý poskytuje služby na efektívny chod gridu. Súčasťou gLite je služba zabezpečujúca monitorovanie úloh prebiehajúcich v gride s názvom Logging & Bookkeeping Service (L&B) [3]. Táto služba umožňuje získať aktuálny stav úlohy a podrobný popis procesu výpočtu úlohy v gridovom prostredí. Užívateľ pristupuje k informáciám zhromaždeným L&B službou prostredníctvom poskytovaného klientského API.

Cieľom tejto práce je navrhnúť grafické užívateľské rozhranie (GUI), ktoré bude komunikovať s L&B službou, získané informácie následne uloží do lokálnej databáze, aby boli prístupné off-line. GUI umožní zhromažďovať dáta z rôznych L&B serverov. Následne dokáže tieto dáta prehľadne zobraziť, dovolí definovať množiny úloh a nad týmito množinami počítať štatistické údaje.

---

[1] http://eu-egee.org

[2] http://glite.web.cern.ch/glite/

[3] http://egee.cesnet.cz/cs/JRA1/LB/

Na začiatku táto práca načrtne problematiku gridových služieb, predstaví motivácie pre tvorbu gridu, presnejšie definuje pojem grid a stručne popíše koncept spolupráce inštitúcii na spoločnom gride. Následne oboznámi čitateľa s históriou projektu EGEE, aktuálnym stavom a so súvisiacim vývojom gLite. Dôraz je kladený na problematiku monitorovania úloh prebiehajúcich v gridovom prostredí. V ďalšej časti je stručne popísaná história vzniku L&B služby. Čitateľ sa dozvie základný koncept L&B služby zahrňujúci generovanie udalostí o jednotlivých úlohách, ich následné zhromažďovanie na príslušnom serveri a adekvátne spracovanie pre potreby koncového užívateľa. Samostatná kapitola je venovaná webovým službám a komunikácii L&B služby s užívateľmi prostredníctvom klientského API založenom práve na tejto technológii. V praktickej časti predstaví základnú charakteristiku implementovaného GUI, ktoré kontaktuje L&B službu, získa požadované dáta a uloží ich do lokálnej databáze. Údaje potrebné na ustanovenie spojenia s L&B službou získava z popisu poskytovaného API, do istej miery tak dokáže reagovať na niektoré zmeny, ktoré môžu nastať pri ďalšom vývoji služby. Aplikácia ďalej poskytuje nástroje na zobrazovanie detailu jednotlivých úloh, vytváranie množín úloh a počítanie štatistických údajov nad definovanými množinami. Zostavenie detailu úlohy a výpočet štatistík sú realizované prostredníctvom zásuvných modulov, užívateľ tak má možnosť implementovať vlastný výpočet štatistík a prispôsobiť prezentáciu požadovaných informácii svojím špecifickým potrebám.

Kapitola 2

Gridové služby

2.1Vznik a vývoj gridových služieb

Spôsob ako riešiť problém nedostatku výpočetnej sily je zefektívniť využívanie už existujúcich zdrojov. Mnoho výkonných počítačov ostáva z rôznych príčin po netriviálne dlhú dobu nečinných. Zber a analýza dát zo zrážok častíc v urýchľovači môže začať až po spustení experimentu, predpoveď dráhy kométy sa dá vypočítať až po presnom zmeraní k tomu potrebných fyzikálnych veličín. Mnohé inštitúcie nárazovo potrebujú zvýšený výpočtový výkon, zdroje zakúpené z tohto dôvodu by po opadnutí špičky ostali nevyťažené, nehovoriac o značných finančných nárokoch takéhoto riešenia. Za určitých okolností sa ukazuje byť výhodnejšie samostatne spravovať zdroje potrebné pre bežnú činnosť a o kapacity nutné na zvládnutie nárazovej záťaže sa deliť s ďalšími spoločnosťami čeliacimi podobnému problému. Spolupracujúce organizácie si tak podelia finančné nároky na získanie a spravovanie daných zdrojov a výrazne sa zlepší aj efektivita ich využitia. Ponúka sa teda myšlienka zaviesť určitú infraštruktúru a vzájomne prepojiť počítače do jedného celku tak, aby boli schopné efektívne spolupracovať. To umožní zdieľať výpočetné aj pamäťové zdroje, čím sa dosiahne potrebný objem výpočetných kapacít.

Práve rozvinutie myšlienky zdieľať výpočetné zdroje stojí za vznikom pojmu grid. Ian Foster a Carl Kesselman v publikácii *The Grid: Blueprint for a New Computing Infrastructure* z roku 1998 predstavili analógiu medzi gridom a elektrickou rozvodnou sieťou (anglicky electrical power grid – z toho názov grid). Na začiatku 20. storočia neexistovali veľké elektrárne. Každý, kto chcel mať elektriku k dispozícii, si musel zaobstarať vlastný generátor. Zavedenie elektriky bolo preto drahé a zároveň dochádzalo k neefektívnemu využívaniu elektrických kapacít. Až zavedením rozsiahlej rozvodnej siete došlo k masívnemu elektrifikovaniu. Táto sieť totiž umožnila efektívne zdieľať výrobné kapacity, čím poskytla relatívne lacný a štandardizovaný prístup k elektrickej energii [1]. Podobne sú v súčasnosti rozdrobené výpočetné zdroje. Každá organizácia spravuje vlastné počítače a súvisiace hardvérové komponenty, neexistuje štandardizovaná možnosť efektívneho zdieľania a koordinácie týchto zdrojov so zdrojmi iných inštitúcií. Získať potrebnú výpočetnú silu, pamäťové úložisko alebo rôzny špecializovaný hardvér je pre jednotlivé spoločnosti finančne náročné. Na druhej strane jediná inštitúcia, ak takými zdrojmi disponuje, ich často nedokáže naplno využiť. Gridová služba by mala poskytnúť hardvérovú a softvérovú infraštruktúru, pomocou ktorej budú takéto zdroje prepojené, zaviesť univerzálne štandardy súvisiace s používaním tejto infraštruktúry tak, aby sa výrazne zefektívnilo využitie jednotlivých zdrojov.

2.2Definícia gridu

V roku 2002 Ian Foster spresnil definíciu gridu zavedením troch podmienok. Ako gridový systém by sa mal označovať systém, ktorý:

•zabezpečuje koordináciu a štandardizovaný prístup k distribuovaným zdrojom, celý systém nepodlieha žiadnej centralizovanej správe.

•používa výhradne voľne dostupné obecné protokoly a rozhrania. Tým je zaručená univerzálna schopnosť gridu integrovať heterogénne prvky.

•grid pracuje takým spôsobom, aby bola zaručená požadovaná kvalita služby, napríklad dostupnosť alebo latencia, umožňuje zdroje koordinovať do tej miery, aby funkčnosť celého gridu bola výrazne vyššia ako možnosti jednotlivých komponent [2].

2.1Kategorizácia gridov

Podľa účelu použitia rozlišujeme rôzne typy gridových systémov:

❖Výpočtové gridy (Computational Grid) – zabezpečujú služby a prostriedky pre vykonávanie náročných výpočtov v distribuovanom prostredí, zahŕňajú výkon veľkého množstva menších výpočetných jednotiek za účelom vytvorenia dostatočnej kapacity pre tieto výpočty. Podľa spôsobu komunikácie, resp. miere nezávislosti jednotlivých úloh delíme výpočtové gridy na rôzne podkategórie:

➢High-performace gridy – používajú sa na riešenie úloh s vysokou mierou závislosti, jednotlivé komponenty gridu spolu komunikujú pomocou rýchlej komunikačnej siete. Grid sa navonok chová ako jeden celok.

➢High-throughput gridy – vhodné sú naopak na množstvo úloh, ktoré sú na sebe pomerne nezávislé, nie je nutná extrémne rýchla komunikácia medzi jednotlivými komponentami gridu. Grid tohto typu je využiteľný pri analýze veľkého množstva nezávislých dát.

➢Volunteer gridy – využívajú dobrovoľníkov, ktorý zapoja svoje, zväčša obyčajné osobné počítače prostredníctvom internetu do gridu. Za predpokladu, že daný počítač je nedostatočne vyťažený aplikáciami dobrovoľníka je grid schopný nevyužitý výkon spotrebvať na svoje vlastné výpočty.[3]

❖Dátové gridy (Data Grid) – poskytujú úložisko pre obrovské objemy dát. Ponúkajú komplexné služby nad zhromaždenými dátami ako napríklad indexovanie, vyhľadávanie alebo katalogizáciu uložených dát.

❖Gridy pre spoluprácu (Collaborative Grid) – vytvárajú prostredie pre spoluprácu, umožňujú vzdialene spravovať jednotlivé hardvérové a softvérové komponenty gridu [4].

2.1Virtuálne organizácie

V súvislosti s rozvojom gridu došlo k vytváraniu tzv. virtuálnych organizácií (VO). Pod týmto pojmom sa rozumie zoskupenie užívateľov, ktorí spolupracujú na určitých výpočtoch prebiehajúcich v gride a preto je nutné, aby mali spoločné práva prístupu k dátam, výpočtovým elementom a ďalším komponentám gridu. VO sú vysoko škálovateľné, v gride môžu existovať malé VO, ktoré tvoria pracovníci jednej inštitúcie zodpovedajúci za výpočet vyžadujúci rozšírené privilégiá v rámci gridu, ktoré nie je vhodné prideliť organizácii ako celku, ako aj VO, do ktorých patrí množstvo pracovníkov viacerých inštitúcii spolupracujúcich na gride. Virtuálne organizácie môžu fungovať na dlhodobej báze, alebo iba v rámci jedného projektu. Po skončení príslušných výpočtov je relatívne jednoduché ukončiť aj spoluprácu v rámci VO [5].

Kapitola 3

Enabling Grids for E-SciencE

3.1História projektu EGEE

Bezprostredným predchodcom projektu Enabling Grids for E-SciencE (EGEE) je projekt DataGrid[4]. Bol financovaný Európskou úniou, motiváciou pre tento projekt boli plánované aktivity medzinárodného ústavu CERN[5] (hlavne urýchľovač častíc LHC[6]) vyžadujúce vysokú mieru paralelizácie a schopnosť získavať, ukladať a analyzovať veľké objemy dát. Projekt trval 3 roky a skončil v máji 2004, za ten čas sa podarilo vytvoriť funkčný základ dátového gridu. Na projekte participovalo mnoho subjektov, jedným z nich bolo aj združenie CESNET[7].

3.1.1Združenie CESNET

CESNET je spoločná aktivita vysokých škôl v ČR a Akadémie vied ČR, toto združenie vzniklo v roku 1996. Hlavným cieľom bolo založiť a spravovať vysokorýchlostnú akademickú sieť v ČR. Táto sieť mala slúžiť na výmenu vedeckých informácií a výskum pokročilých sieťových technológií. Neskôr sa CESNET zameral aj na výskum a vývoj gridových služieb, v súvislosti s týmto smerom výskumu bol do združenia začlenený projekt MetaCentrum[8].

MetaCentrum zahájilo svoju činnosť v roku 1996 v súvislosti s nákupom výkonných počítačov pre niektoré vysoké školy v ČR. Tieto počítače nedokázali spolu efektívne komunikovať a koordinovať svoju činnosť. Odstránenie tohto nedostatku sa stalo jednou z prvých úloh MetaCentra. Postupom času sa MetaCentrum prirodzene začalo venovať výskumu gridových služieb. V súčasnosti je jeho činnosť primárne venovaná rozvoju národného gridu, výskumu na poli gridov a podieľa sa na niekoľkých európskych projektoch súvisiacich s gridovými službami. Do združenia CESNET bolo MetaCentrum začlenené v roku 1999.

3.1.2Projekt EGEE

Na výsledky a skúsenosti projektu DataGrid nadviazal ďalší európsky projekt EGEE, prostredníctvom združenia CESNET na projekte participuje aj Česká republika. Prvá fáza projektu, dnes označovaná ako EGEE-I, začala v marci 2004 a bola koncipovaná na dva roky. Pôvodný plný názov znel Enabling Grids for E-Science in Europe, avšak posledné dve slová boli

[4]http://eu-datagrid.web.cern.ch/eu-datagrid
[5]http://web.cern.ch/public/
[6]http://lhc.web.cern.ch/lhc
[7]http://www.cesnet.cz
[8]http://meta.cesnet.cz/cs/

z názvu čoskoro vypustené, ako sa do projektu zapájali inštitúcie mimo región Európy. Úlohou tejto fázy bolo nadviazať na projekt DataGrid, rozšíriť existujúci grid o ďalšie zdroje, prispôsobiť grid nielen na výpočty súvisiace s projektmi CERNu, ale na ľubovoľné úlohy z ďalších oblastí vedy a výskumu, s čím súvisí aj hľadanie nových partnerov z ďalších vedeckých oborov vyžadujúcich značné výpočetné zdroje. Základná sieťová infraštruktúra gridu je tvorená európskou výskumnou sieťou GÉANT2[9]. Spočiatku bol grid založený na middleware z projektu DataGrid, ale súčasne došlo k vývoju novej strednej vrstvy nazvanej gLite, ktorá sa čoskoro stala základom gridu.

Bezprostredne po skončení prvej fázy začala druhá fáza EGEE-II ktorá bola primárne zameraná na rozširovanie gridu a zlepšovanie servisných a sieťových aktivít. Zaoberala sa efektívnym spôsobom riadenia a spravovania komplexného gridu, zdokonaľovaním užívateľskej podpory, vysokorýchlostným pripojením nových komponent a získavaním nových partnerov. Pokračoval však aj výskum a vývoj gLite, najmä v oblasti zaistení kvality služieb, bezpečnosti a možností testovania [7]. V súčasnosti prebieha už tretia fáza projektu, EGEE-III. Termín ukončenia je stanovený na 30. 5. 2010. Hlavným cieľom je transformovať grid na udržateľný výpočetný systém, ktorý pre svojich užívateľov garantuje kvalitu služby. Základ utvoria jednotlivé gridové aktivity zapojených krajín (národné gridy), ktoré budú ďalej prepojené do celoeurópskeho gridu. Po skončení EGEE-III prevezme kontrolu nad správou gridu European Grid Initiative[10] (EGI), ktorá bude koordinovať národné gridové aktivity z pohľadu celého gridu, zabezpečí tak dozor a efektívne fungovanie gridu ako celku.

V súčasnosti EGEE zahŕňa vyše 260 rôznych stanovíšť v 55 krajinách sveta. Využíva výpočtovú silu 150 tisíc procesorov, ktorá je prístupná 24 hodín denne. Grid poskytuje 28 PB diskovej kapacity a 41 PB páskovej pamäti na dlhodobé uloženie dát. Prístup ku gridu má 140 registrovaných virtuálnych organizácií, registrovaných užívateľov je približne 140 tisíc. Priemerný počet úloh spracovaných v gride je 330 tisíc za jeden deň.
(zdroj: http://www.eu-egee.org/, december 2009)

3.2Middleware gLite

Middleware gLite [8] je vyvíjaný v rámci projektu EGEE, vznikol kompletným prepracovaním strednej vrstvy vyvinutej v rámci projektu DataGrid. Oficiálnym middleware projektu EGEE je od mája 2006, teda takmer od úplného začiatku celého projektu. Pri vývoji gLite boli využité koncepty a skúsenosti aj z ďalších projektov zaoberajúcich sa gridovými službami, najmä LCG[11], VDT[12], Globus Toolkit [9] a Condor[13]. Služby gLite tvoria rozhranie medzi užívate-

[9]http://www.geant2.net/
[10]http://web.eu-egi.eu/
[11]http://lcg.web.cern.ch/lcg/
[12]http://vdt.cs.wisc.edu/
[13]http://www.cs.wisc.edu/condor/

ľom a gridom. Na spodnej úrovni gLite zabezpečuje kooperáciu medzi rôznorodými komponentami gridu, umožňuje unifikovaný spôsob zapojenia nových komponent a koordináciu ich činnosti [10]. Na druhej strane gLite tiež poskytuje služby vyššej úrovne. Umožňuje autentizáciu a autorizáciu prístupu k jednotlivým gridovým zdrojom, dovoľuje užívateľovi predkladať úlohy do gridu, zobrazovať výstupy skončených úloh, rušiť úlohy, manipulovať so súbormi v gride a mnohé ďalšie funkcie potrebné pre plnohodnotnú prácu v gridovom prostredí.

Jednotlivé úlohy v gride založenom na gLite prebiehajú na logickej jednotke zvanej Computing Element (CE). Každý CE sa skladá z brány (Grid Gate), ktorá slúži ako vstupný bod CE, z lokálnej správy zdrojov, ktoré CE obsahuje (Local Resource Management System) a zo samotných výpočetných uzlov (Worker Nodes), na ktorých sa realizujú konkrétne úlohy. Služby, ktoré gLite poskytuje nie sú implementované ako komplexný nedeliteľný celok, práve naopak, gLite sa skladá z množstva menších modulov. Užívateľ tak získava možnosť prispôsobiť si gLite na mieru a užívať iba služby, ktoré sú relevantné pre jeho špecifické potreby [11].

Middleware gLite je postavený na architektúre orientujúcej sa na služby (SOA). Systémy postavené na tejto architektúre sa vyznačujú množstvom dostupných služieb, ktoré sú riadne definované, samostatné a nezávislé na výsledku iných služieb. Preto je možné ich opakovane používať v rôznych kontextoch. Jednotlivé komponenty tohto systému sú iba voľne previazané a pomocou jednotného rozhrania volajú služby iných komponent. Takáto architektúra zásadne zjednodušuje interoperabilitu gLite s inými gridovými službami a umožní kompatibilitu s budúcimi štandardami, napríklad s Web Service Resource Framework[14] alebo Open Grid Service Architecture[15] [10]. Podporované platformy pre gLite sú Scientific Linux verzie 4 alebo 5 a Debian 4.

3.3Monitorovanie úloh v gride založenom na gLite

Dôležitou súčasťou funkčnosti gridových služieb je sledovanie a archivácia jednotlivých úloh. Úlohy v gride migrujú medzi rôznymi komponentami, v rozsiahlych distribuovaných systémoch je monitorovanie úloh pomerne náročný problém. Sledovanie úloh je pritom nutné pre potreby analýzy efektívneho využitia gridu, vytvárania štatistík prevádzky a fungovania gridu, zistenia kompletného stavu úloh alebo na opakované spustenie zrušenej alebo nedokončenej úlohy. Práve za vývoj služieb zabezpečujúcich sledovanie a archiváciu úloh zodpovedá združenie CESNET. Na účely monitoringu výpočetných úloh sa v rámci gLite používa služba Logging and Bookkeeping (L&B).

Archiváciu dokončených úloh v prostredí gLite zabezpečuje služba Job Provenance (JP). Ukladá kompletný stav úlohy, výstupné dáta a celý proces jej realizácie (migrácie úlohy medzi jednotlivými komponentami gridu počas výpočtu), tu JP aktívne spolupracuje s L&B.

[14]http://www.globus.org/wsrf
[15]http://www.globus.org/ogsa

Okrem toho JP uchováva aj vstupné dáta a ďalšie informácie o kontexte úlohy potrebné pre opakované spustenie úlohy za rovnakých podmienok. Všetky tieto dáta musia byť uložené dlhodobo, z archívu JP sa dáta o úlohách neodstraňujú. JP umožňuje k uloženým dátam pripájať anotácie užívateľov, ktoré pomáhajú zlepšovať organizáciu dát a dajú sa využiť napríklad pri vyhľadávaní. Archivované informácie sa dajú s výhodou použiť napríklad pri opakovaní experimentov alebo pri príprave ďalších podobných experimentov [12].

Kapitola 4

Logging and Bookkeeping služba

Služba L&B [12] bola vytvorená v rámci projektu DataGrid, jej vývoj pokračoval vo všetkých fázach projektu EGEE v súvislosti z rozvojom gLite middleware, ktorý absorboval L&B ako svoju samostatnú súčasť. K vývoju tejto služby významnou mierou prispelo združenie CESNET.

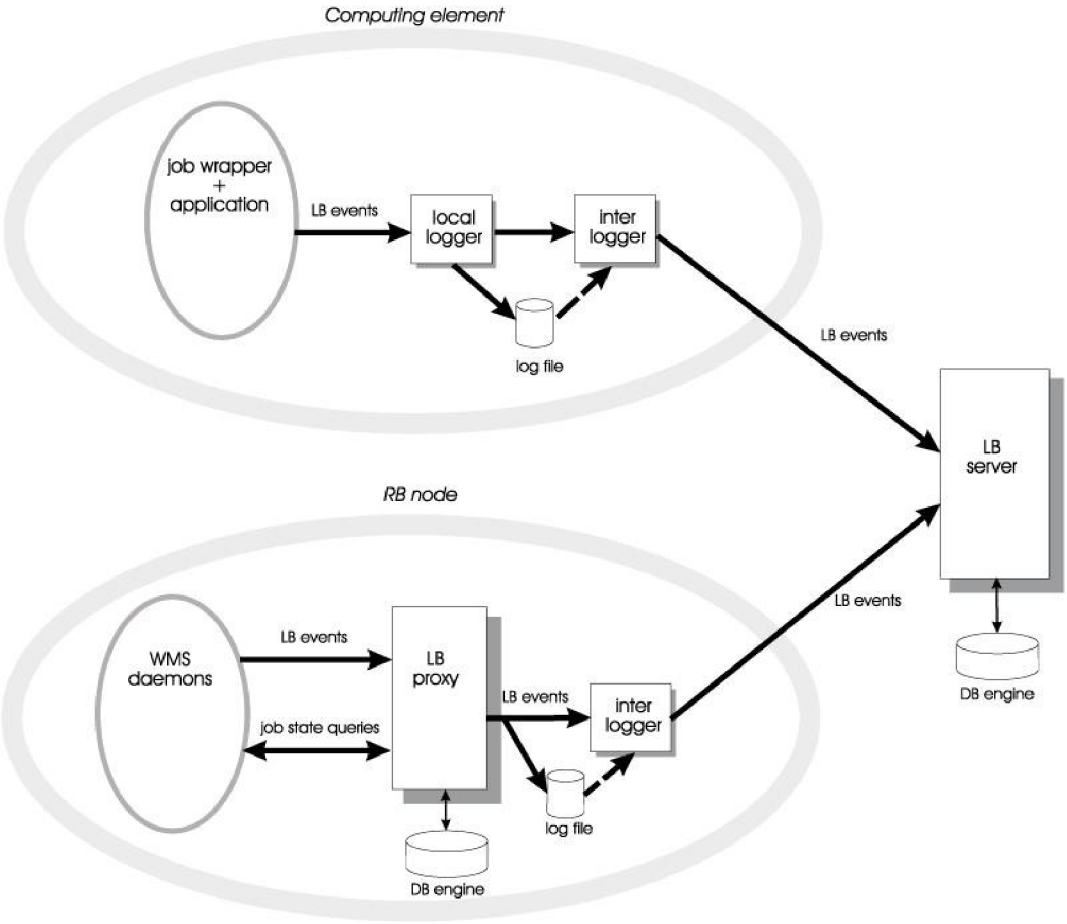
Workload Management System (WMS) je systém pre správu záťaže. V gride slúži na distribúciu úloh medzi jednotlivými komponentami tak, aby nedochádzalo k neúmernému preťaženiu jednej komponenty na úkor iných komponent za predpokladu, že tieto komponenty sú schopné poskytnúť analogické služby. Pôvodne bola L&B súčasťou WMS, postupom času však došlo ku jej osamostatneniu. Nezávislosť služby zjednodušuje komunikáciu s ďalšími službami operujúcimi s úlohami a uľahčuje rozširovanie služby o monitoring ďalších typov úloh (nielen výpočetných) prebiehajúcich v gride, ako sú napríklad dátové prenosy. Tiež sa výrazne zjednodušuje väčšina služieb WMS (nemusia uchovávať a aktualizovať stavy úloh, môžu fungovať ako bezstavové), čím sa zrýchľuje sa správa záťaže, čo je hlavná náplň činnosti WMS.

Základnou monitorovacou jednotkou z pohľadu L&B je úloha (job). Na rozlíšenie jednotlivých úloh každá úloha dostane pri svojom predložení do gridu jedinečný globálny identifikátor (*jobid*). Všetky informácie o konkrétnej úlohe sa zhromažďujú a ukladajú do jednej databáze. Užívateľ má možnosť získať kompletný status svojej úlohy a nemusí skúmať logy jednotlivých komponent, ktoré danú úlohu spracovávali. Nehľadiac na to, že k týmto logom užívateľ nemusí mať prístup, prípadne ani nemusí vedieť, že existujú. Informácie sú uchovávané ešte nejaký čas po zaznamenaní ukončenia úlohy. To užívateľovi dovoľuje spätne zhodnotiť priebeh výpočtu úlohy a analyzovať prípadné neštandardné správanie.

4.1Jobid a jeho funkcia

V rámci celého gridu existuje množstvo L&B serverov, čo je pochopiteľné vzhľadom na značný počet úloh realizovaných na gride. Jeden server by sa stal úzkym miestom celého gridu a bol by neúmerne preťažovaný v porovnaní s ostatnými komponentami. Viacero L&B serverov však prináša problém, ako determinovať, ktorá úloha sa monitoruje na ktorom servery. Vytvorenie globálnej služby, ktorá bude adresovať úlohu na konkrétny server je len presunutie úzkeho miesta zo samotného serveru na túto službu. Informácia o tom, kde sa zhromažďujú udalosti o danej úlohe je preto obsiahnutá priamo v jobid. Pre komponenty gridu ako aj pre užívateľov je jednoduché získať adresu príslušného L&B servera bez nutnosti volania akejkoľvek vzdialenej služby.

Jobid sa skladá z URL počítača a portu, na ktorom beží L&B databáza, za oddeľujúcim znakom "/" sa nachádza unikátny reťazec, ktorý vznikol z náhodného čísla, časovej známky (timestamp), identifikátoru procesu, ktorý danú úlohu generuje a IP adresy stroja, na ktorom sa daný jobid vytvára. Z týchto údajov sa vytvorí MD5 hash, ktorý je následne zakódovaný funkciou base64. Tento spôsob vytvárania identifikátorov nedáva úplnú istotu unikátnosti, avšak pravdepodobnosť zhody jobid pre dve rôzne úlohy je zmysluplne malá. Príklad jobid úlohy, ktorej prislúcha L&B server na adrese harad.ics.muni.cz, načúvajúci na porte 9250:
`https://harad.ics.muni.cz:9250/-ALaIfXmKywnxq3TLPAB_w`



Obrázok 4.1.: Základný koncept služby L&B [12]

4.2Zhromažďovanie udalostí

Zaznamenávanie informácií o úlohe je realizované prostredníctvom udalostí (event). WMS komponenta generuje udalosť o konkrétnej úlohe vždy, keď nastane určitý definovaný okamih v procese výpočtu. Medzi takéto významné udalosti patrí napríklad transfer úlohy na inú komponentu gridu, akceptovanie alebo odmietnutie úlohy danou komponentou, vloženie

respektíve odstránenie úlohy z fronty komponenty, prerušenie alebo obnovenie výpočtu úlohy. Služba L&B poskytuje API a súvisiace knižnice, ktoré umožňujú generovať udalosti o úlohe. Jednotlivé komponenty majú priamy prístup k tomuto API, nedochádza tak k zbytočnému zaťažovaniu WMS. Generovanie týchto udalostí nesmie mať vplyv na plynulosť výpočtu úlohy, preto je L&B služba založená na asynchrónnom modeli zasielania správ. Komponenta nemusí čakať na potvrdenie prijatia udalosti a pokračuje vo výpočte. Asynchrónnosť zasielania správ však spôsobuje, že L&B služba môže dočasne poskytovať nekonzistentné údaje, pretože istý čas je potrebný k tomu, aby daná udalosť bola spracovaná a ovplyvnila aktuálny status úlohy zaznamenaný v L&B.

4.2.1Local logger a inter logger

Každá WMS komponenta má k dispozícii local logger, ktorý je fyzicky blízko, aby sa eliminovali chyby pri prenose. Udalosti sa na tento logger posielajú synchrónne, naspäť sa odosielajú potvrdenia o prijatí. Local logger udržuje samostatný súbor s udalosťami pre každú úlohu, ktorá sa spracováva na príslušnej komponente.

Local logger následne predá udalosti inter loggeru, ktorý rozpozná cieľový L&B server jednotlivých udalostí, pre každý server vytvorí frontu a postupne rozposiela jednotlivé udalosti, až kým nedostane potvrdenie o prijatí. V prípade akýchkoľvek chýb je inter logger schopný obnoviť svoju činnosť na základe dát z local loggeru. Úspešne doručené správy sú z inter loggeru zmazané.

4.2.2L&B Proxy

Samotný WMS niekedy potrebuje pre svoju činnosť poznať aktuálny stav úlohy. Týmito informáciami však priamo nedisponuje, musel by ich získať prostredníctvom L&B, to značí zavolať vzdialenú službu. Takéto riešenie nie je vhodné, pretože akékoľvek volanie vzdialenej služby je potenciálne časovo náročné a je náchylné na chyby sieťovej infraštruktúry gridu. Na spoluprácu s L&B namiesto toho WMS využíva L&B proxy. Proxy zabezpečuje funkcionalitu local loggeru a navyše poskytuje aj niektoré základné funkcie samotnej L&B služby s nasledujúcimi obmedzeniami:

- stav úlohy je odvodený iba z lokálnych udalostí, preto nemusí byť nutne kompletný
- ku proxy sa pristupuje cez lokálny unixový soket namiesto sieťového rozhrania
- nerealizuje sa autorizácia užívateľov, práva je nutné nastaviť na úrovni unixového soketu
- dáta sú vymazané okamžite po prijatí udalosti oznamujúcej ukončenie úlohy

•nekontroluje sa indexovanie jednotlivých atribútov, nepredpokladá sa výrazný nárast objemu dát v databáze z dôvodu okamžitého mazania dát po skončení úlohy

4.1Formát udalostí

Pre rôzne významné okamihy v procese výpočtu úlohy sú definované rôzne typy udalostí. Udalosť sa skladá z párov atribút = hodnota, jednotlivé typy udalostí sa líšia v množine povinných a nepovinných atribútov. Existuje spoločná báza atribútov, ktoré sú povinné pre všetky typy udalostí. Jeden z nich je jobid, niektoré ďalšie sú sekvenčný kód udalosti (L&B pomocou neho určuje poradie udalosti v rámci ostatných udalostí o danej úlohe), časová známka a identifikátor komponenty generujúcej danú udalosť.

4.2Radenie udalostí

Použitie loggerov zabezpečuje, že udalosti budú prijaté L&B serverom, ale nezaručuje správne poradie príjmu jednotlivých udalostí. Správne poradie udalostí je pritom zásadná znalosť potrebná na odvodenie aktuálneho stavu úlohy. Už pomerne jednoduchá situácia, keď úloha zlyhá na niektorom CE gridu a WMS ju deleguje na iný CE, kde skončí úspešne, môže znemožniť správne odvodenie stavu úlohy. Ak správy dorazia na L&B v obrátenom poradí, posledná udalosť o úlohe z pohľadu L&B naznačuje, že výpočet zlyhal. L&B na účely radenia udalostí využíva sekvenčný kód, jeho hodnota sa predáva medzi jednotlivými komponentami gridu spolu s riadením celej úlohy. Kód sa skladá z hierarchicky usporiadaných čítačov, pri prechode každá gLite komponenta zvýši hodnotu svojho čítača v rámci sekvenčného kódu. WMS v rámci kódu disponuje vlastným čítačom, ktorý je v hierarchii vyššie, ako čítač pre CE. Na základe čítača pre WMS je L&B služba schopná rozpoznať, v akom poradí bola úloha spracovávaná na jednotlivých CE. Hierarchický kód dáva dostatok informácií pre správne zoradenie jednotlivých udalostí, pretože súčasná implementácia gLite zaručuje, že jediné miesto, kde môže nastať vetvenie úlohy (napríklad transfer alebo opakované spustenie) je WMS. Systém loggerov s frontami zabezpečuje, že udalosti od jedného CE prídu na L&B v správnom poradí.

4.3L&B server

L&B server jednotlivé udalosti autentizuje, uloží a spracuje. Samotné udalosti poskytujú komplexný prehľad o stave úlohy, pre užívateľa však je takáto informácia ťažko čitateľná. Server preto z prijatých udalostí odvodí a zosumarizuje status úlohy do prehľadnejšieho formátu, ktorý prezentuje užívateľom. Jednotlivé udalosti sa však neodstraňujú z dôvodu možnej potreby podrobnejšej analýzy. Udalosti sú úmyselne navrhnuté tak, aby poskytovali čo najviac redundantných údajov. Pri transfere úlohy medzi jednotlivými komponentami je v rámci L&B služby vygenerovaných hneď niekoľko udalostí rôznych typov, napríklad udalosť označujúca začiatok transferu, koniec transferu alebo udalosť oznamujúca prijatie respektíve odmietnutie úlohy na cieľovom CE. Takáto redundantná informácia pomáha L&B serveru pri odvodzovaní

statusu úlohy a umožňuje vypočítať tento stav aj v prípade oneskorených, poprehadzovaných alebo dokonca chýbajúcich udalostí.

4.4Bezpečnosť a kontrola prístupu

Informácie o stave úlohy môžu obsahovať citlivé dáta, napríklad niektoré údaje o vlastníkovi úlohy. Rôzne dôverné informácie sa môžu objaviť napríklad pri vyhodnocovaní medicínskych dát. Jednotlivé udalosti prichádzajúce na L&B server musia byť spoľahlivo autentizované, aby sa zabránilo ich podvrhu. Všetky L&B komponenty preto používajú výhradne zabezpečené kanály. Na autentizáciu a ustanovenie šifrovaného spojenia sa používa protokol TLS, ktorý využíva X. 509 certifikáty operujúce s asymetrickými šiframi. Každá komponenta L&B disponuje takýmto certifikátom a dokáže si ustanoviť spoločné šifrované spojenie s ľubovoľnou inou L&B komponentou. Asymetrické šifrovanie je možné zároveň použiť aj ako autentizačný mechanizmus. Pri použití klientského API na získanie informácií o úlohách je užívateľ tiež povinný preukázať svoju identitu prostredníctvom certifikátu.

Štandardne má prístup k informáciám o danej úlohe iba jej vlastník (ten, kto ju predložil do gridu). Každá úloha má svoj vlastný zoznam určujúci entity, ktoré majú prístup k dátam o danej úlohe (Access Control List – ACL). Tento zoznam špecifikuje užívateľov, ktorým bude povolené získavať informácie z L&B servera o danej úlohe. ACL je pod výhradnou kontrolou vlastníka, ktorý môže nastaviť práva prístupu konkrétnym užívateľom alebo celej skupine užívateľov (napríklad všetkým členom VO[16]), alebo explicitne tieto práva odobrať konkrétnemu užívateľovi, aj keď je členom dôveryhodnej VO.

4.5Aktuálny stav

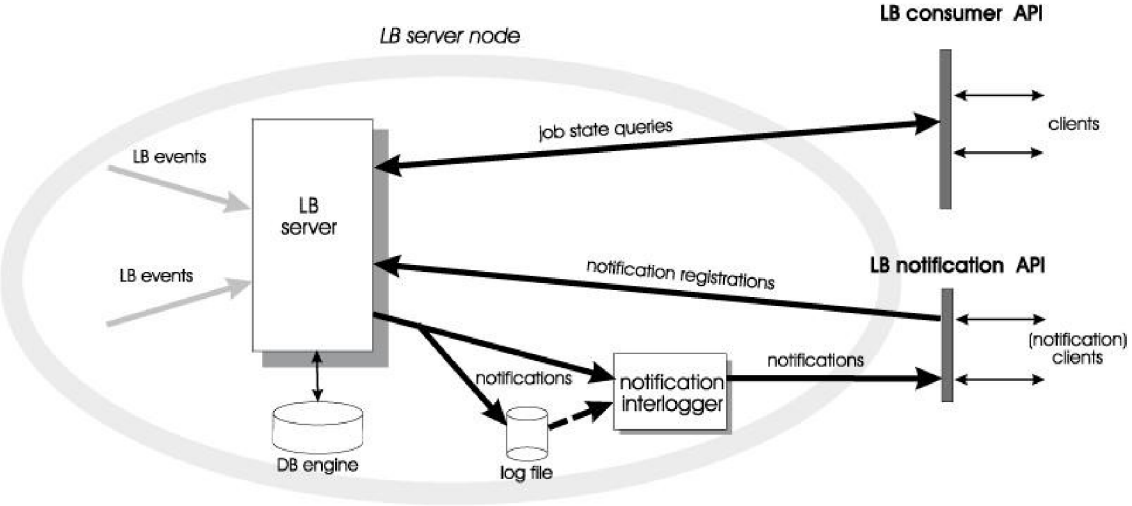
V rámci gLite 3.1 sa používa L&B 1.x. Vývoj tejto verzie je už ukončený. Pre gLite 3.2 sa vyvíja L&B 2.0, zatiaľ iba ako experimentálna služba. Ostatné verzie sú v súčasnosti považované za zastarané. L&B služba je pomerne portabilná, funkčnosť bola vyskúšaná na platformách Debian 4, Scientific Linux 5, Scientific Linux CERN 4, SuSE 9, 10 a 11, v 32 aj 64 bitových verziách. L&B nepoužíva IP adresy v žiadnom internom procese, preto by nemal byť problém ani s kompatibilitou s IPv6 (zdroj: http://glite.web.cern.ch/glite/, december 2009).

[16] Virtuálna organizácia

Kapitola 5

Užívateľský prístup k L&B službe

Užívatelia pristupujú k dátam L&B služby prostredníctvom synchrónneho API, ktoré umožňuje klásť dotazy na jednotlivé úlohy. Existujú dva typy dotazov, prvý poskytuje údaje o statuse úlohy alebo množine úloh, druhý údaje o jednotlivých udalostiach. Zvyčajne stačí prvý typ, druhý slúži na skúmanie neštandardného správania, kde je potrebný nízkoúrovňový prístup k jednotlivým udalostiam.



Obrázok 5.1.: Interakcia užívateľov so službou L&B [12].

Existujú dve implementácie klientského API, jedna využíva jazyk C/C++, klient pristupujúci k L&B prostredníctvom tohto rozhrania využíva knižnice napísane v tomto jazyku. Druhé API je založená na webových službách [13], ktoré sú platformovo nezávislé a nekladú na klienta žiadne obmedzenia, čo sa týka implementácie. Ako nadstavba nad API využívajúcom webové služby bolo vytvorené API implementované v jazyku Java [14]. Toto API uľahčuje komunikáciu programov napísaných v jazyku Java s L&B službou tým, že poskytuje predpripravené triedy pre vzdialené volanie webových služieb. Okrem dotazovacieho API L&B poskytuje aj API pre notifikácie. Užívateľ sa môže zaregistrovať a špecifikovať podmienky, za ktorých mu majú byť notifikácie posielané. Týmto sa znižuje záťaž L&B servera, pretože užívatelia nemusia periodicky zisťovať stav svojich úloh, ale server pri zmene stavu úlohy automaticky pošle užívateľovi notifikáciu.

Užívateľské dotazy sú prevedené na SQL dotazy, ktoré vrátia požadované informácie z databáze. Klientské API však neposkytuje plnú silu jazyka SQL, pretože niektoré dotazy by

mohli v prípade rozsiahlych dát uložených v databáze vyvolať príliš veľkú záťaž. Povolené sú iba dotazy, na výpočet ktorých je možné použiť už existujúci a dostatočne selektívny databázový index.

5.1Metódy klientského API používajúceho webové služby

Pre potreby tejto práce je najdôležitejšie klientské API založené na webových službách (v ďalšom texte iba klientské alebo užívateľské API). Toto API poskytuje päť metód prístupu k L&B serveru:

•GetVersion – metóda vráti verziu L&B služby, vstupný argument je reťazec, ktorý však metóda nepoužíva, je prítomný kvôli spätnej kompatibilite

•UserJobs – získa všetky úlohy užívateľa, ktorý ju zavolal, je bezparametrická

•JobStatus – zistí kompletný stav zadanej úlohy, na vstupe vyžaduje jobid úlohy a zoznam príznakov (flag), ktoré určujú objem informácií obsiahnutých v statuse úlohy

•QueryJobs – vráti status úloh vyhovujúcich zadanej podmienke, opäť sa ako vstupný argument objavuje zoznam príznakov definujúcich mieru podrobností statusu úlohy

•QueryEvents – vráti informácie o požadovaných udalostiach

Užívateľské API definuje 5 rôznych príznakov určujúcich, ktoré informácie budú zahrnuté do získaných statusov jednotlivých úloh:

•CLASSADS – súčasťou statusu bude aj dlhý popis úlohy

•CHILDREN – vráti sa zoznam podúloh danej úlohy

•CHILDSTAT – získa sa kompletný status podúloh danej úlohy, tj. zadané príznaky sa rekurzívne aplikujú na podúlohy

•CHILDHIST_FAST – získa sa histogram stavov jednotlivých podúloh, použije sa rýchly výpočet, niektoré stavy môžu ostať neznáme

•CHILDHIST_THOROUGH – vráti sa plný histogram stavov jednotlivých podúloh

Metóda QueryJobs vyžaduje ako vstupný argument množinu podmienok definujúcich istú skupinu úloh, o ktoré má užívateľ záujem. Na zápis týchto podmienok L&B používa jednoduchý dotazovací jazyk. Schéma zápisu jednoduchej podmienky je nasledovná:

$atr_1\ OP_1\ value_1 \quad \wedge \quad ... \quad \wedge \quad atr_n\ OP_n\ value_n$

Zložená podmienka sa skladá z konjunkcie disjunkcií. Schematické znázornenie zloženej podmienky vyzerá takto:

$$
\begin{aligned}
&(\mathrm{attr}_1\ \mathrm{OP}\ \mathrm{hodnota}_{1.1} \vee \ldots \vee \mathrm{attr}_1\ \mathrm{OP}\ \mathrm{hodnota}_{1.i_1}) \wedge \\
&\quad (\mathrm{attr}_2\ \mathrm{OP}\ \mathrm{hodnota}_{2.1} \vee \ldots \vee \mathrm{attr}_2\ \mathrm{OP}\ \mathrm{hodnota}_{2.i_2}) \wedge \\
&\qquad \ldots \\
&\quad \wedge (\mathrm{attr}_n\ \mathrm{OP}\ \mathrm{hodnota}_{n.1} \vee \ldots \vee \mathrm{attr}_n\ \mathrm{OP}\ \mathrm{hodnota}_{n.i_n})
\end{aligned}
$$

5.1Webové služby

Webové služby [13] sú technológia umožňujúca komunikáciu medzi navzájom nezávislými komponentami prostredníctvom siete. Využívajú výlučne otvorené štandardy a protokoly, čo ich robí nezávislé na komerčných produktoch a univerzálne použiteľné pre všetky systémy. Pri využití webových služieb dochádza k prenosu XML správ medzi komunikujúcimi uzlami. Presný formát týchto správ definuje protokol SOAP[17]. Na prenos správ je obecne možné využiť akýkoľvek otvorený sieťový protokol, v praxi webové služby využívajú najčastejšie HTTP, respektíve HTTPS.

Typické volanie webovej služby pozostáva z niekoľkých krokov. Konzument najprv vytvorí SOAP správu, ktorá identifikuje konkrétnu službu a kóduje príslušné vstupné parametre. Táto správa je následne doručená pomocou daného sieťového protokolu producentovi (cieľový server poskytujúci službu). Producent spracuje prijatú správu, zavolá požadovanú službu s príslušnými vstupnými parametrami, výsledok služby opäť zakóduje do SOAP správy, ktorú pošle naspäť konzumentovi.

Aby konzument vedel zostaviť SOAP správu, musí poznať presné umiestnenie a definíciu vstupných parametrov služby. Tiež potrebuje vedieť vnútornú štruktúru výsledku, ktorý daná služba vráti, aby ho dokázal náležite spracovať. Túto znalosť poskytuje producent služby. Pre tieto účely vznikol jazyk WSDL[18]. Je založený na XML a umožňuje producentovi univerzálnym spôsobom popísať ním poskytované služby. Popis založený na WSDL má výhodu aj v tom, že je možné ho strojovo spracovať.

[17] www.w3.org/TR/soap
[18] www.w3.org/TR/wsdl

Kapitola 6

Grafické užívateľské rozhranie

V tejto časti bližšie predstavíme základný koncept a štruktúru grafického užívateľského rozhrania (GUI) vyvinutého v rámci tejto práce. Stručne povedané, GUI poskytuje grafické prostredie na formuláciu dotazov a pomocou webových služieb jednotlivé dotazy odosiela na príslušné L&B servery. Získaný výsledok (celkový status úlohy alebo množiny úloh) uloží do lokálnej databáze, z ktorej dokáže dáta znova načítať a zobraziť. Rozhranie umožňuje pracovať v off-line režime a zadávať komplexné dotazy na výber úloh z lokálnej databáze zapísané prostredníctvom jazyka SQL. Podporované sú iba dotazy na úlohy, prácu s udalosťami GUI neumožňuje.

Implementácia je realizovaná v jazyku Java SE 6, ktorý funguje na platformách MS Windows aj Unix/Linux. Súčasťou tohto jazyka je knižnica Swing, ktorá ponúka podporu pre programovanie grafických aplikácií. Na parsovanie WSDL popisu aplikácia používa balíček dom4j[19]. Pri implementácii bolo použité vývojové prostredie Netbeans, ktoré obsahuje WYSIWYG editor pre tvorbu grafických prvkov Matisse.

Služba L&B je stále vo vývoji, GUI je preto navrhnuté tak, aby dokázalo reagovať na niektoré zmeny klientského API, najmä čo sa týka vnútornej štruktúry prijatých dát. Status úlohy, ktorý vracia L&B služba tvorí súbor rozličných parametrov rôznych typov a vlastností, GUI dokáže pružne reagovať na zmeny očakávané v ďalšom vývoji, ako je pridanie alebo odstránenie niektorých parametrov, zmena typu parametra alebo zmena počtu hodnôt, ktoré môže daný parameter nadobúdať. Schéma tabuliek používaných na uloženie statusu úloh sa odvodí od WSDL popisu služby, implementácia metód podieľajúcich sa na ukladaní a načítaní jednotlivých statusov tiež využíva informácie odvodené z tohto popisu, čím sa dosahuje požadovaná flexibilita. Pri zmene klientského API L&B služby stačí aplikácii zadať nový WSDL súbor.

Jedna z výhod GUI oproti už existujúcim rozhraniam je schopnosť naraz prehliadať dáta z viacerých L&B serverov. Môže však nastať situácia, keď verzie L&B služby budú natoľko rozdielne, že štruktúra tabuliek vygenerovaných podľa popisu jednej L&B služby bude nevyhovujúca pre dáta druhej služby. Riešením tohto stavu je meniť lokálnu databázu podľa toho, ku ktorým L&B službám chce užívateľ pristupovať. Nie vždy je ale vhodné vytvárať nové databáze, preto ako jeden z nastaviteľných vstupných parametrov GUI je prefix pre názvy tabuliek. Rozhranie vytvára a používa iba tie tabuľky, ktoré začínajú daným prefixom. Ani tento spôsob nerieši problém, ako prostredníctvom GUI naraz pristupovať k dátam príliš rozdielnych L&B služieb, ale dovoľuje všetky tieto dáta uložiť do jednej databáze.

[19] www.dom4j.org/

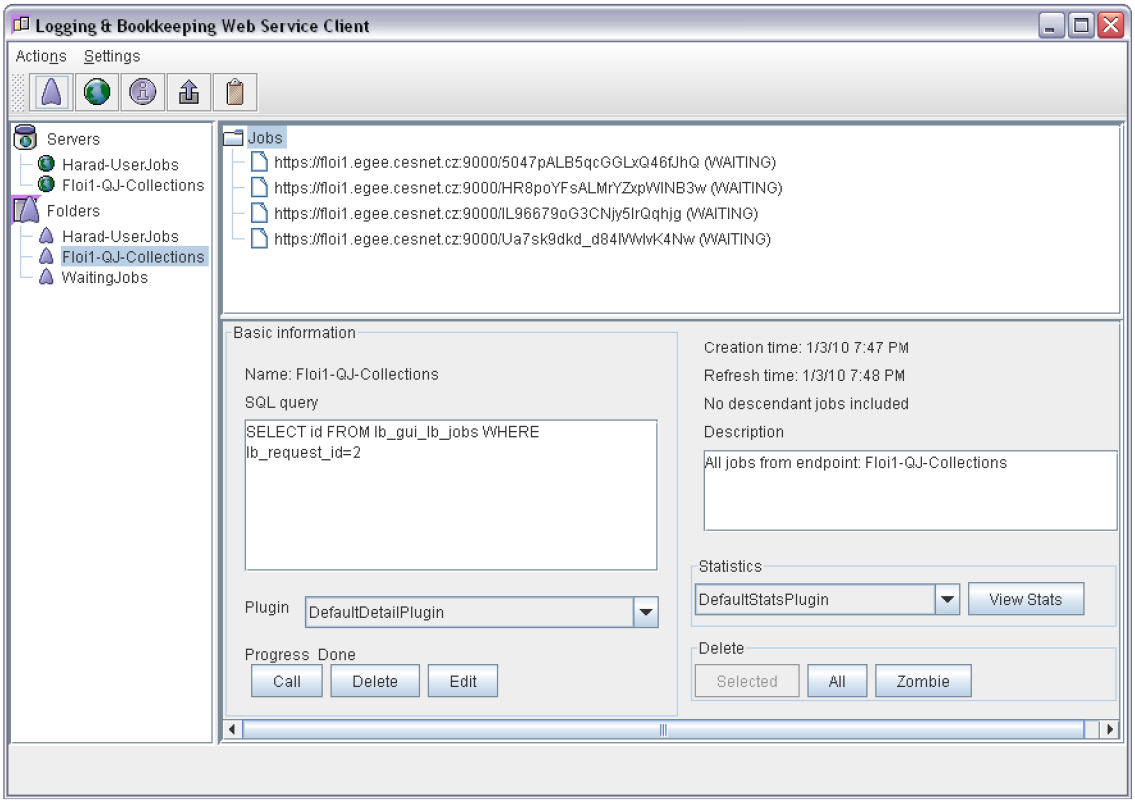
Aplikácia umožňuje užívateľom rozšíriť funkcionalitu pomocou troch typov zásuvných modulov (plugin). Prvý typ pluginu definuje zobrazenie statusu jednej úlohy (viz 6.10), druhý umožňuje zhromažďovať rôzne štatistické údaje nad zadanými množinami úloh (viz 6.9) a tretí typ slúži na implementáciu konzumenta webovej služby, ktorý na spodnej vrstve komunikuje s L&B serverom a obdržaný výsledok ukladá do lokálnej databáze (viz 6.12).

Zhromažďovanie dát uložených na viacerých L&B serveroch môže mať za následok vytvorenie relatívne rozsiahlej lokálnej databáze obsahujúcej tisíce úloh . Najčastejšia práca s aplikáciou bude off-line (nájdenie požadovaných úloh, načítanie detailu úlohy alebo počítanie štatistík nad množinami úloh), ktorá vyžaduje najmä rýchle vyhľadanie potrebných údajov. Aplikácia preto poskytuje užívateľovi jednoduché rozhranie na administráciu indexov nad tabuľkami, do ktorých ukladá údaje o úlohách (viz 6.11), ktoré dokážu výrazne zrýchliť práve takýto typ SQL dotazov.

Implementácia GUI sa skladá z niekoľkých samostatných knižníc a programov. Základný program predstavuje implementáciu samotného GUI. Tri knižnice definujú obecné rozhrania pre jednotlivé typy zásuvných modulov. Ďalšia knižnica obsahuje triedy, ktoré sa využívajú v pluginoch pre konzumenta webovej služby a aj v samotnom GUI. Aby sa pri implementácii vyhlo ich duplicitnej definícii, boli vyčlenené do samostatnej knižnice (viz 6.6). Posledná knižnica obsahuje triedu `JobDetail` (viz 6.4), ktorá prestavuje flexibilnú implementáciu statusu úlohy.

6.1Použitie GUI

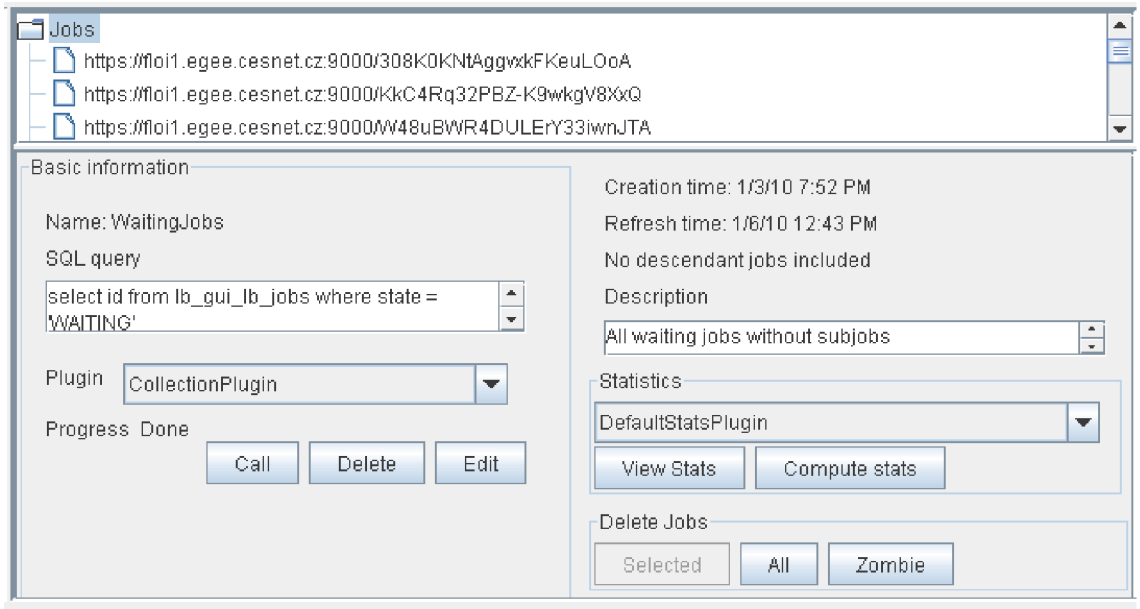
Základné rozloženie ovládacích prvkov GUI v mnohom pripomína osvedčený výzor typického e-mailového klienta. Úvodné okno aplikácie je vertikálne rozdelené na užšiu ľavú časť a širšiu pravú časť. V ľavej časti sa nachádza zoznam zložiek. Je tvorený stromovou štruktúrou, ako prvé sú zobrazené zložky reprezentujúce spojenia na konkrétny L&B server (ďalej tiež nazývané endpointy, viz 6.1.1). V druhej časti zoznamu sa nachádzajú zložky definujúce množiny úloh vytvorené z dát uložených v lokálnej databáze (tzv. virtuálne zložky alebo foldre, viz 6.2). Tieto umožňujú prezerať jednotlivé úlohy a počítať rôzne štatistiky o príslušnej množine úloh.



Obrázok 6.1: Základné rozloženie grafických prvkov GUI

Obsah pravej časti sa líši podľa typu zložky, ktorý je označený v zozname v ľavej časti. V prípade virtuálnej zložky je pravá časť podobne ako pri e-mailovom klientovi rozdelená horizontálne na dve samostatné podčasti. Vrchná polovica slúži na zobrazenie množiny úloh, ktoré patria do príslušného foldru. Tento zoznam pozostáva z jobid jednotlivých úloh, po kliknutí na konkrétny jobid sa v spodnej polovici zobrazí detail označenej úlohy. Pri dvojnásobnom

kliknutí sa tento detail navyše ukáže v novom okne. Ak v zozname úloh nie je označená žiadna konkrétna úloha, v spodnej časti sú vyobrazené všeobecné informácie o danej virtuálnej zložke.



Obrázok 6.2: Základné informácie o virtuálnej zložke

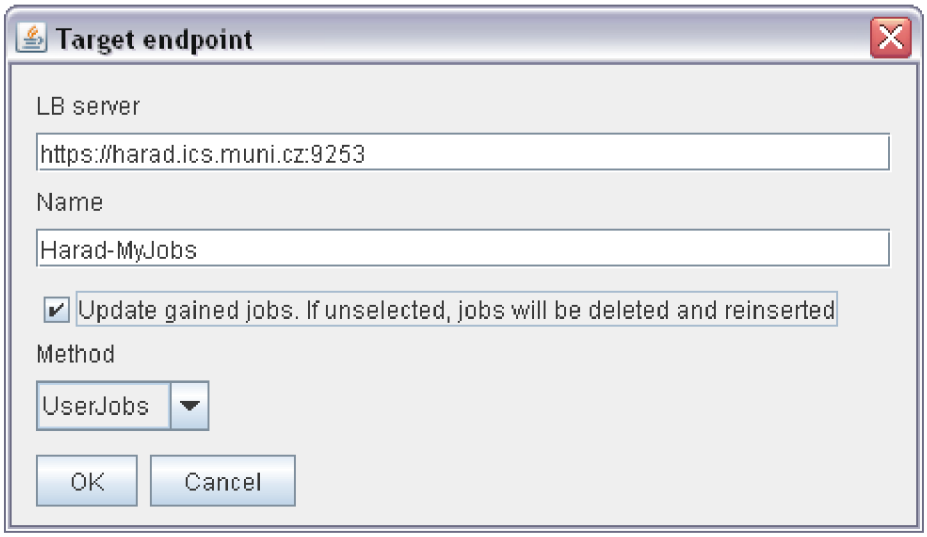
V prípade endpointu pravá časť obsahuje všetky údaje týkajúce sa príslušného spojenia na L&B server a volanej metódy klientského API.

Obrázok 6.3: Pravá strana obsahuje panel s informáciami o spojení na L&B server

6.1.1Ukladanie úloh pre prácu off-line

Prvý krok pri používaní rozhrania je získanie údajov o požadovaných úlohách prostredníctvom spomínaných endpointov. GUI umožňuje špecifikovať presnú adresu cieľového servera

a konkrétnu metódu spolu so vstupnými parametrami, ktorá bude zavolaná pomocou webových služieb. V ďalšom kroku musí užívateľ špecifikovať ako sa bude dané spojenie správať pri aktualizácii. Prvá možnosť je zmazať jednotlivé záznamy o úlohách a uložiť ich nanovo. Druhá možnosť je, že sa stávajúce záznamy aktualizujú čerstvo získanými údajmi (viz. 6.12.2). Trieda, inštancie ktorej zabaľujú tieto informácie, sa nazýva jednoducho `Endpoint`. Aplikácia dovoľuje vytvárať viacero takýchto spojení na jeden L&B server, užívateľ má možnosť získať dáta o úlohách z jedného servera prostredníctvom viacerých dotazov. Inštancie triedy `Endpoint` neumožňujú prístup k detailom o jednotlivých úlohách. Všetky údaje z rozličných L&B serverov uložia do príslušných tabuliek, pre prístup k týmto úlohám GUI využíva virtuálne zložky.

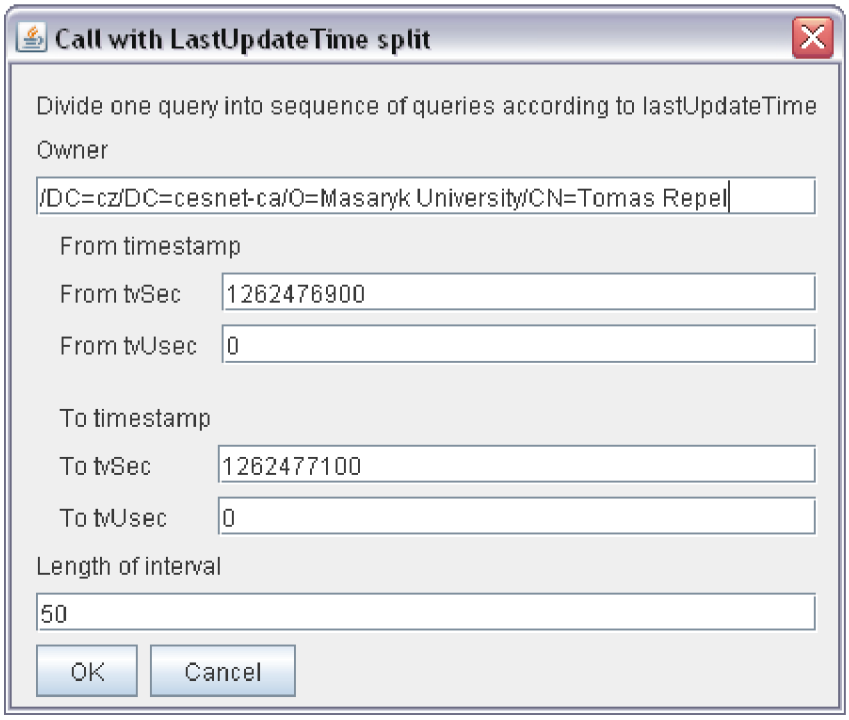


Obrázok 6.4.: Vytvorenie nového spojenia na L&B server

6.1.2Rozdelenie dotazu podľa časového razítka

V prípade príliš obecného dotazu alebo pri značnom počte úloh evidovaných L&B službou sa do prijatého výsledku zahrnie viac úloh, ako je aplikácia schopná spracovať. Spracovanie príliš objemných výsledkov vyžaduje viac pamäte, ako si môže maximálne alokovať Java Virtual Machine (JVM), v rámci ktorej bežia Java aplikácie. Pri vzniku tejto situácie aplikácia na toto užívateľa upozorní a ponúkne možnosť rozdeliť dotaz na sekvenciu viacerých dotazov podľa parametru LASTUPDATETIME. Tento parameter je povinný a obsahuje časové razítko, kedy naposledy L&B služba aktualizovala status úlohy. Užívateľ zadá počiatočný čas a koncový čas, ktoré ohraničujú časový úsek, v ktorom musela byť úloha naposledy aktualizovaná. Očakáva sa počet uplynulých sekúnd od začaitku unixovej epochy (1.1.1970) alebo dátum vo formáte *yyyy-MM-dd HH:mm:ss*. Nakoniec užívateľ nastaví dĺžku intervalov, na ktoré sa rozseká časový úsek determinovaný počiatočným a koncovým časom. Do špecifikovaného dotazu sa pre každý

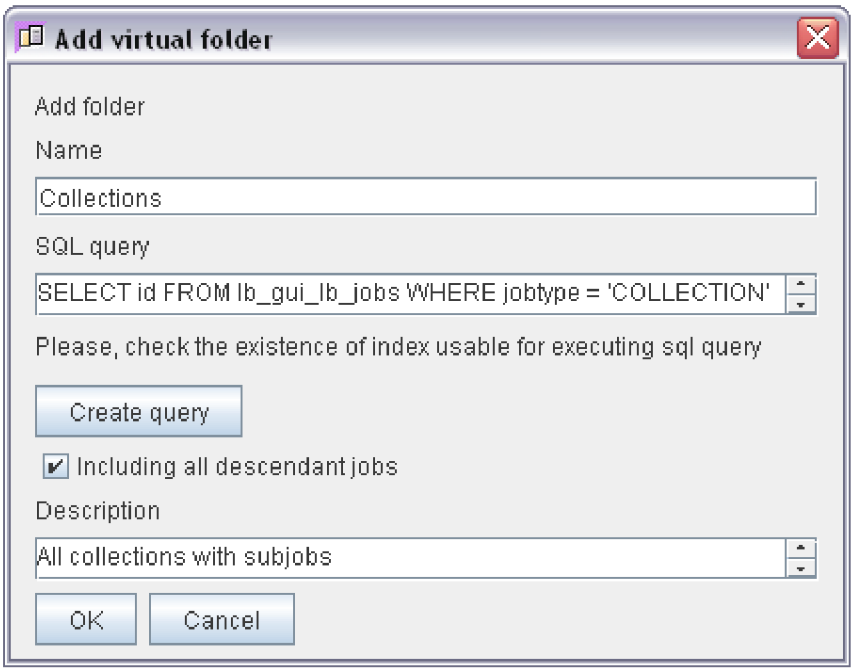
vzniknutý interval pridá podmienka vyžadujúca, aby parameter LASTUPDATETIME obsahoval hodnotu patriaciu do daného intervalu. Dotaz sa tak rozloží na sekvenciu viacerých dotazov, ktoré už je možné spracovať a uložiť. Všetky takto získané úlohy patria pod jedno spojenie. Takýto postup je možný pre metódy UserJobs a QueryJobs. Drobný rozdiel spočíva v tom, že pri metóde UserJobs navyše užívateľ zadá vlastníka úloh. Táto hodnota je však predvyplnená podľa užívateľovho certifikátu.



Obrázok 6.5.: Rozdelenie dotazu podľa LASTUPDATETIME

6.2Koncept virtuálnych zložiek

Virtuálna zložka pracuje nad lokálnou databázou, jej základom je SQL dotaz, ktorý určuje úlohy patriace do zložky. Hlavný účel je selektovať tie úlohy, ktorých status (viz. 6.4) užívateľa zaujíma, respektíve definovať množinu úloh, nad ktorou sa budú počítať štatistické údaje. Zložka dovoľuje nastaviť konkrétny zásuvný modul na zobrazenie kompletného statusu úlohy. Trieda reprezentujúca virtuálnu zložku sa nazýva `Folder` a je súčasťou implementácie samotného GUI. Aplikácia tiež poskytuje grafické rozhranie, pomocou ktorého si užívateľ môže vytvoriť jednoduchú SQL podmienku (viz. 6.2.3).



Obrázok 6.6: Vytvorenie virtuálnej zložky

Pri zobrazení virtuálnej zložky sa postupne načítajú všetky jej úlohy a zobrazí sa ich jobid. Po kliknutí na konkrétny jobid sa načítajú údaje o danej úlohe a prostredníctvom zvoleného pluginu sa zobrazí detail úlohy. Pri úlohách, ktoré už majú načítaný kompletný detail sa v zozname zobrazí nielen ich jobid, ale aj stav danej úlohy. K zoznamu je priradené kontextové menu obsahujúce jedinú položku, ktorá vyvoláva vyhľadávacie dialógové okno. Užívateľ zadá časť jobId požadovanej úlohy, ktorá sa v zozname vyhľadá, aplikácia posunie zoznam úloh tak, aby bola daná úloha viditeľná a zobrazí príslušný detail úlohy. Pri vytvorení endpointu aplikácia automaticky vytvorí aj virtuálnu zložku, ktorá združuje všetky úlohy stiahnuté a uložené v rámci tohto spojenia.

L&B monitoruje aktuálne úlohy spracovávané gridovým prostredím, po uplynutí istej doby od ukončenia úlohy sa údaje z L&B servera zmažú. Novšie verzie služby si však permanentne uchovajú aj jobid všetkých starších úloh, ostatné informácie odstránia a úlohe priradia stav PURGED (tzv. zombie jobs). Ak aplikácia po načítaní detailu úlohy zistí, že sa jedná o takúto úlohu, popri jobid a stave úlohy sa v zozname zobrazí aj nápis ZOMBIE.

6.2.1Mazanie úloh prostredníctvom virtuálnych zložiek

Pri zobrazení detailu o zložke a detailu úlohy má užívateľ k dispozícii tlačítka, ktoré dovoľujú zmazať všetky úlohy danej zložky, zmazať všetky označené úlohy z úloh zložky a zma-

zať všetky *zombie* úlohy. Získa tak možnosť ručne zmazať nepotrebné úlohy, ktoré sú navyše alebo skresľujú počítané štatistiky. Záznamy o úlohách sa odstránia z lokálnej databáze, pre ich prípadné znovuuloženie je nutné aktualizovať endpoint, v rámci ktorého sa dané úlohy vložili do databáze.

6.2.2Spracovanie dotazu virtuálnej zložky

Dotaz zadaný v rámci zložky sa pri selektovaní nepoužíva priamo. Aplikácia z neho extrahuje SQL podmienku, ktorú použije na načítanie identifikátorov vyhovujúcich úloh. V prípade, že zložka má obsahovať aj všetky podúlohy, aplikácia z lokálnej databáze rekurzívne získa príslušné identifikátory. Zhromaždené dáta GUI použije na zostavenie zoznamu úloh zložky.

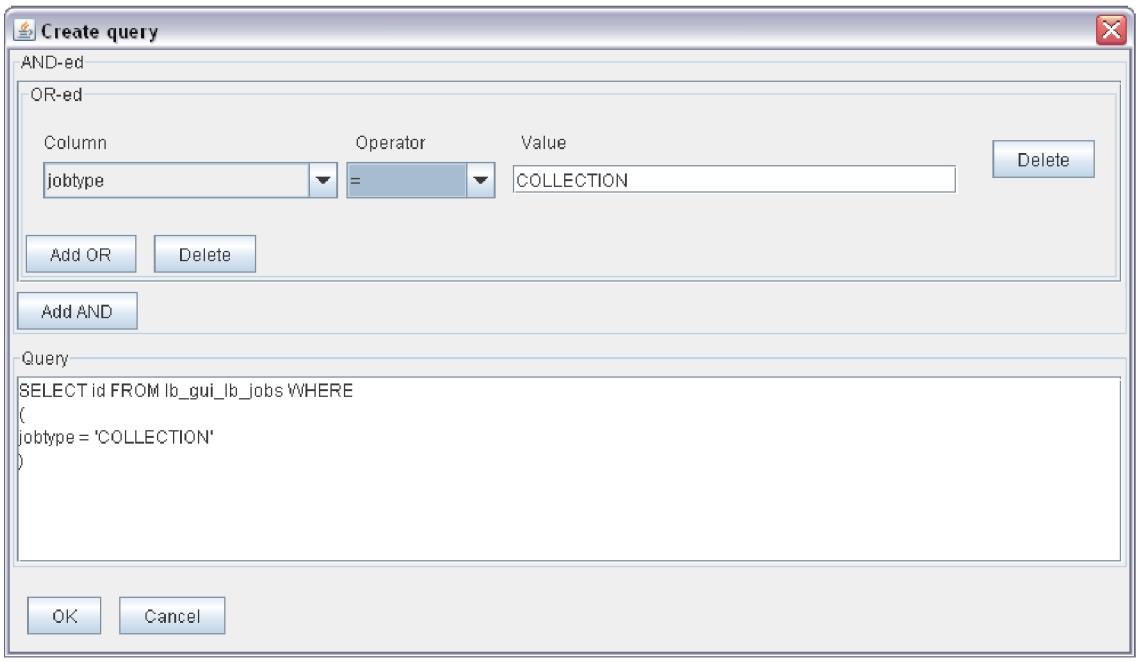
Takáto práca s dotazom zložky dovoľuje užívateľovi klásť jednoduchšie dotazy. Pri špecifikovaní dotazu nemusí spájať (left join) tabuľku lb_jobs s tabuľkami uchovávajúcimi viachodnotové parametre za predpokladu, že ním vytváraná podmienka s týmito parametrami neoperuje. Ak podmienka obsahuje nejaký viachodnotový parameter, lb_jobs sa spojí iba s tabuľkou príslušného parametra. Jednoduchý dotaz na všetky práve bežiace úlohy môže byť napríklad takýto:

```
SELECT id FROM lb_jobs WHERE state = 'RUNNING'
```

Príznak určí, či do zložky budú zahrnuté aj všetky podúlohy bežiacich úloh (nutne nemusia byť v stave RUNNING).

6.2.3Vytvorenie SQL podmienky pre virtuálnu zložku

Na uľahčenie zadávania SQL podmienky aplikácia poskytuje grafické rozhranie pre tvorbu jednoduchých podmienok v tvare konjunkcie disjunkcií. V prípade tvorby podmienky prostredníctvom tohto rozhrania aplikácia obecne v klauzuli FROM spojí tabuľku lb_jobs so všetkými tabuľkami pre viachodnotové parametre. Užívateľ by mal v záujme urýchlenia databázových operácii nepotrebné tabuľky z klauzule FROM odstrániť. Dialógové okno slúžiace na vytvorenie vlastného dotazu obsahuje tlačítko na pridávanie AND-častí, v rámci každej tejto časti je možné pridať OR-časti, v ktorej užívateľ špecifikuje parameter, operátor a požadovanú hodnotu. Počet AND a OR-častí nie je obmedzený. V dolnej časti okna sa postupne ukazuje výsledný generovaný dotaz.



Obrázok 6.7: Vytvorenie SQL podmienky pre virtuálnu zložku

6.3Vstupná konfigurácia

Vstupné údaje sa zadávajú prostredníctvom súboru `config.ini` umiestnenom priamo v adresári aplikácie. Je to obyčajný textový súbor, dáta sú uložené vo formáte kľúč = hodnota. Každý takýto pár predstavuje jeden riadok (viz Príloha A). Riadok začínajúci dvojicou obyčajných lomítok je považovaný za komentár. Jednotlivé kľúče, ku ktorým GUI načíta príslušné hodnoty sú:

•mysql_db – lokálna databáza, do ktorej sa ukladajú údaje o úlohách

•db_login – užívateľské meno pre prístup do lokálnej databáze

•db_password – heslo pre prístup do lokálnej databáze

•table_prefix – prefix pre všetky názvy tabuliek, ktoré sa vytvoria pre potreby aplikácie v danej databáze

•log_path – cesta k súboru, do ktorého sa ukladajú logovacie správy

•user_cert – cesta k certifikátu užívateľa, podporovaný formát je PEM

•user_cert_password – heslo k súboru s kľúčom pre certifikát

•user_cert_key – cesta k súboru s kľúčom pre certifikát vo formáte PEM

•ca_files – cesta k certifikátom dôveryhodných certifikačných autorít

•wsdl_path – cesta k WSDL popisu klientského API L&B služby

•plugin_dir – cesta k adresári, v ktorom sú uložené pluginy zodpovedajúce za zobrazenie detailu úlohy

•stats_plugin_dir – cesta k adresári, v ktorom sú uložené pluginy pre výpočet štatistík nad definovanými množinami úloh

•timeout – časový limit pre konzumenta webovej služby

Nie je nutné heslá zadávať priamo do súboru, aplikácia pri absencii týchto parametrov dokáže pred ich použitím vyzvať užívateľa, aby potrebné heslo zadal prostredníctvom dialógového okna. Ak sa kľúč pre heslo v `config.ini` nachádza, ale hodnota je prázdna, aplikácia to interpretuje ako zadané prázdne heslo.

6.1Trieda JobDetail

Táto trieda implementuje celkový status úlohy. Tento status je tvorený rôznymi parametrami, trieda je navrhnutá tak, aby bola schopná reprezentovať statusy úloh s odlišnou vnútornou štruktúrou. Aplikácia tak je schopná pracovať s mierne rozdielnymi verziami L&B služby a jej klientského API. Odlišnosti v štruktúre statusu úlohy, ktorým sa trieda JobDetail dokáže prispôsobiť sa týkajú najmä pridávania a odstraňovania parametrov zo statusu úlohy, zmeny typu parametrov alebo predefinovania parametra z jednohodnotového na viachodnotový parameter a naopak. Existujú automatizované nástroje, napr. AXIS2[20], ktoré sú schopné na základe WSDL popisu štruktúry statusu úlohy vytvoriť triedu, ktorá predstavuje príslušnú implementáciu. Použitím takto generovanej triedy nie je možné zaručiť požadovanú flexibilitu aplikácie voči prípadným zmenám klientského API.

Presná XML štruktúra prijatých dát je aplikácii známa z WSDL popisu. Na uloženie jednohodnotových parametrov (napr. vlastník úlohy), sa používa mapa, ktorá jednotlivým kľúčom priradí konkrétnu hodnotu. Hodnoty všetkých jednoduchých typov sú kódované do textových reťazcov (trieda `String`). Kľúč tvorí názov XML elementu v SOAP správe, ktorý danú hodnotu obsahoval. Podobný princíp je použitý pri viachodnotových parametroch, jediný roz-

[20] http://ws.apache.org/axis2/

diel spočíva v tom, že k príslušnému kľúču je priradená množina hodnôt. Pri pridaní parametru sa pridá príslušný záznam do mapy. Odstránenie parametru z WSDL popisu spôsobí, že záznam s daným kľúčom sa do mapy vôbec nevloží. Nevýhoda z tohto vyplývajúca je, že trieda `JobDetail` neposkytuje klasické get/set-metódy. V prijatom výsledku máme napríklad informáciu o stave úlohy `<state>DONE</state>.` Na získanie tohto stavu neexistuje metóda `getState()`, ale iba obecná get-metóda, ktorej sa zadá príslušný kľúč a výsledkom je požadovaná hodnota: `getSimpleProperty("state")`.

V prípade zložených typov sa tieto rekurzívne rozložia na typy jednoduché, ktoré je možné priamočiaro reprezentovať ako textový reťazec. V tomto prípade kľúč k jednotlivým hodnotám tvoria postupne názvy všetkých XML elementov oddelených podtržítkom (znak _), ktoré obsahujú príslušnú hodnotu. Súčasťou statusu úlohy môže byť napríklad informácia, vyjadrujúca čas, kedy sa daná úloha dostala do aktuálneho stavu:

```
<stateEnterTime>
  <tvSec>1252679641</tvSec>
  <tvUsec>316686</tvUsec>
</stateEnterTime>
```

Jednotlivé hodnoty sa do inštancie triedy JobDetail uložia pod kľúčmi stateEnterTime_tvSec a stateEnterTime_tvUsec.

L&B služba umožňuje užívateľom pridávať anotácie k jednotlivým úlohám. Pre tieto anotácie trieda JobDetail udržuje samostatnú mapu a príslušné get/set-metódy. Ďalšiu množinu udržuje pre podúlohy danej úlohy. Trieda je rekurzívne definovaná, prvky tejto množiny sú opäť inštancie triedy `JobDetail`. Okrem údajov získaných priamo z L&B servera trieda udržuje aj informácie súvisiace s interným spracovaním danej úlohy v rámci aplikácie. Obsahuje premennú id reprezentujúcu identifikátor úlohy, ktorý jej priradila databáza pri uložení, identifikátor endpointu, ktorý vrátil výsledok obsahujúci túto úlohu a dátum poslednej aktualizácie úlohy v lokálnej databáze. Nakoniec trieda obsahuje statickú metódu, ktorá je schopná vytvoriť a inicializovať objekt tejto triedy s použitím dát uložených v lokálnej databáze.

6.2Uloženie stavu úlohy do lokálnej databáze

Aplikácia predpokladá, že užívateľ disponuje dostatočnými oprávneniami na vytváranie nových tabuliek. Schéma tabuliek určených pre ukladanie statusu úloh sa odvodí podobným spôsobom, ako sa tvoria kľúče parametrov pri vytváraní inštancie triedy `JobDetail`. Do hlavnej tabuľky nazvanej lb_jobs sa ukladajú jednohodnotové parametre, pričom názov jednotlivých stĺpcov tabuľky je totožný s kľúčmi daných hodnôt, ako boli vytvorené v triede `JobDetail`. Okrem takto vytvorených stĺpcov obsahuje tabuľka stĺpec s primárnym indexom nazvaný id, stĺ-

pec s URL adresou a portom zdrojového L&B servera, stĺpec pre identifikátor endpointu, ktorého výsledok obsahuje danú úlohu a stĺpec uchovávajúci dátum poslednej aktualizácie úlohy.

6.2.1 Spracovanie viachodnotových parametrov

Pre viachodnotové parametre sa vytvárajú samostatné tabuľky, ktoré majú stĺpec resp. stĺpce, do ktorých sa ukladá hodnota príslušného parametra, stĺpec obsahujúci primárny index a stĺpec, ktorý obsahuje databázový identifikátor úlohy, ku ktorej daná hodnota patrí. Názvy týchto tabuliek obsahujú okrem globálneho prefixu zhodného pre všetky tabuľky aj vlastný prefix *attr_*. Názvy tabuliek a jednotlivých stĺpcov na ukladanie hodnoty parametra sú opäť odvodené z WSDL popisu. Status úlohy môže obsahovať niekoľko anotácii, ktoré k danej úlohe priradili užívatelia. Jednotlivé anotácie sú uložené v XML elemente nazvanom userTags. Typ anotácie (vnútorná štruktúra hodnoty elementu userTags) je v WSDL popise definovaný takto:

```
<xsd:complexType name="tagValue">
  <xsd:sequence>
    <xsd:element name="tag" type="xsd:string" />
    <xsd:element name="value" type="xsd:string" />
  </xsd:sequence>
</xsd:complexType>
```

Príslušná tabuľka, ktorú aplikácia na základe týchto informácii vytvorí na ukladanie jednotlivých anotácii sa nazve attr_usertags a bude obsahovať štyri stĺpce. Prvé dva stĺpce sú spoločné pre všetky tabuľky viachodnotových parametrov, nezávisia na WSDL. Stĺpec s názvom id obsahuje primárny index a druhý, nazvaný job_id identifikuje úlohu, ktorej je daná anotácia priradená. Ďalšie dva stĺpce sú nazvané usertags_tag a usertags_value, vychádzajú z typu anotácie definovaného vo WSDL popise a obsahujú konkrétnu anotáciu.

6.2.2 Uloženie špeciálnych parametrov

Existujú dva parametre, k ukladaniu ktorých pristupuje aplikácia špeciálnym spôsobom. Nazývajú sa childrenHist a stateEnterTimes. Oba tieto parametre sú zloženého typu a oba sú viachodnotové. Hodnoty parametra childrenHist reprezentujú histogram stavov podúloh danej úlohy, teda počty podúloh danej úlohy nachádzajúcich sa v jednotlivých stavoch. Hodnoty parametra stateEnterTimes obsahujú časy, v ktorých sa daná úloha dostala do konkrétnych stavov. Typ týchto parametrov je zložený z dvoch jednoduchých typov, prvý typ v oboch prípadoch obsahuje reťazec reprezentujúci stav úlohy. Druhý jednoduchý typ v prípade parametra childrenHist predstavuje číslo reprezentujúce príslušný počet podúloh v danom stave. V prípade parametra stateEnterTime druhý typ predstavuje dátum, kedy sa úloha do daného stavu dostala. Počet hodnôt oboch parametrov môže maximálne dosiahnuť počtu všetkých možných stavov, ktoré rozlišuje gLite middleware. Znalosť maximálneho počtu hodnôt dovoľuje pristupovať

k týmto parametrom ako k jednohodnotovým. Aplikácia parsovaním WSDL zistí množinu možných stavov úloh a pre každý stav vytvorí jeden stĺpec v tabuľke lb_jobs pre oba parametre. Názov jedného z možných stavov je DONE, stĺpec pre parameter childrenHist s hodnotou obsahujúcou stav DONE bude nazvaný childrenHist_DONE_count. Pre parameter stateEnterTimes sa príslušný stĺpec nazve stateEnterTimes_DONE_time. Štandardne by aplikácia vytvorila pre tieto typy samostatnú tabuľku so stĺpcami, do ktorých by uložila stav a príslušnú druhú hodnotu zloženého typu. Výhoda tohto postupu je v tom, že pri SQL dotaze obsahujúcom podmienku týkajúcu sa týchto parametrov nie je nutné spájať žiadne tabuľky, pretože údaje sa nachádzajú priamo v lb_jobs. Výrazne sa tak zníži záťaž databáze, čo je dôležité vzhľadom na očakávaný častý výskyt týchto parametrov v dotazoch užívateľov. Nevýhodou je, že pri ukladaní statusu úlohy do lokálnej databáze je nutné brať ohľad na toto špeciálne chovanie, čím sa mierne komplikuje implementácia zásuvného modulu pre konzumenta webovej služby, ktorý toto ukladanie zabezpečuje.

6.3Knižnica obecných tried

Táto knižnica obsahuje triedy potrebné pre GUI a súčasne aj pre zásuvné moduly implementujúce konzumenta webovej služby, ktorú poskytuje L&B server. V tejto knižnici sa nachádza napríklad už spomínaná trieda `Endpoint`, ktorá definuje spojenie na L&B server.

6.3.1Trieda WsdlParser

Trieda `WsdlParser` obsahuje metódy, ktoré slúžia na získavanie informácií z WSDL popisu služby. Všetky metódy sú deklarované ako statické, pretože informácie z WSDL popisu sú potrebné v mnohých iných triedach. Nie je nutné inštanciu tejto triedy komplikovane predávať ako parameter, ani opakovane vytvárať nový objekt. Práve táto trieda parsovaním WSDL popisu konštruuje SQL dotazy na vytvorenie databázových tabuliek, získava množinu prípustných operátorov a operandov pre tvorbu podmienok v dotazovacom jazyku L&B služby, definuje zoznam stavov, ktoré rozlišuje gLite middleware alebo napríklad určuje množinu povolených príznakov pre vstup do metód JobStatus a QueryJobs.

6.3.2Trieda QueryCondition

Trieda `QueryCondition` reprezentuje zloženú podmienku v dotazovacom jazyku L&B služby. Zoznam takýchto podmienok je vstupný argument pre metódu QueryJobs. Objekt tejto triedy definuje konkrétny parameter, ktorého sa podmienka týka. Samotné podmienky, ktoré daný parameter musí splňovať reprezentuje trieda `QueryRecord`, každá inštancia `QueryCondition` môže obsahovať viacero objektov typu `QueryRecord`, tie sú potom navzájom spojené logickou spojkou OR. Jedna inštancia `QueryCondition` tak predstavuje disjunkciu tvrdení dotazovacieho jazyka L&B služby vzťahujúcu sa na konkrétny parameter úlohy.

6.4Databázová schéma

Informácie o virtuálnych zložkách, spojeniach na L&B server a štatistikách sa ukladajú do lokálnej databáze. Aplikácia pri spustení skontroluje, či sa príslušné tabuľky v databáze nachádzajú. Ak nie, automaticky ich vytvorí. Pri reálnom behu aplikácie majú všetky názvy tabuliek spoločný konfigurovateľný prefix, v ďalšom texte budeme uvádzať iba nemennú časť názvu tabuliek.

Okrem tabuliek pre viachodnotové parametre statusu úlohy, ktoré závisia na WSDL a tabuliek pre zásuvné moduly na výpočet štatistík GUI vytvára nasledovné tabuľky:

- lb_request
- query_condition
- query_record
- query_rec_value
- flag
- job_id
- folder
- last_update_time_split
- children_db_id
- lb_jobs

Do tabuľky lb_request sa ukladajú užívateľom definované spojenia na L&B server (z dát obsiahnutých v tejto tabuľke sa inicializujú inštancie triedy `Endpoint`). Do jednotlivých stĺpcov sa uloží názov spojenia, URL servera, kľúč metódy klientského API, čas vytvorenia a čas poslednej obnovy.

Tabuľky query_condition, query_record a query_rec_value slúžia na uloženie dát reprezentujúcich podmienky dotazovacieho jazyka L&B služby. Tabuľka flag definuje zoznam príznakov určujúcich mieru požadovaných podrobností v detaile úlohy. Job_id uchováva vstupný identifikátor úlohy pre metódu JobStatus. Všetky tieto tabuľky slúžia na ukladanie vstupných

argumentov pre metódy klientského API, query_condition, flag, job_id obsahujú cudzí kľúč, ktorý špecifikuje, ku ktorému endpointu daný argument patrí.

Užívateľom vytvorené virtuálne zložky sa uchovávajú v tabuľke folder. Pri spustení aplikácie sa z týchto dát vytvorí zoznam inštancií triedy `Folder`. Jednotlivé stĺpce obsahujú názov zložky, SQL dotaz na úlohy patriace do zložky, príznak, či do zložky treba zaradiť aj potomkov daných úloh, identifikátor endpointu, ak zložka k nejakému patrí, názvy zásuvných modulov na zobrazenie detailu a štatistík, popis zložky, čas vzniku a čas posledného obnovenia.

V prípade rozdelenia dotazu podľa parametra LASTUPDATETIME sa relevantné údaje uložia do tabuľky last_update_time_split. Aj v prípade zadania hraníc sledovaného intervalu prostredníctvom dátumu sa tieto údaje uložia ako počty sekúnd od začiatku unixovej epochy.

Pri uložení úlohy databáza priradí tejto úlohe ďalší unikátny identifikátor. Tabuľka children_db_id uchováva informáciu o podúlohách jednotlivých úloh prostredníctvom týchto identifikátorov. Obsahuje stĺpec job_id do ktorého uloží identifikátor úlohy, do ďalšieho stĺpca child_id sa uloží databázovýidentifikátor podúlohy.

Posledná tabuľka, ktorú GUI vytvára je lb_jobs, ukladá sa do nej kompletný status úloh. Jej štruktúra z väčšej časti závisí na WSDL popise. Aplikácia tam pridá stĺpec s primárnym indexom id, stĺpec s URL adresou L&B servera, stĺpec obsahujúci cudzí kľúč na konkrétny endpoint, ktorý zodpovedá za stiahnutie danej úlohy a stĺpec uchovávajúci dátum poslednej aktualizácie úlohy v rámci lokálnej databáze.

6.1Ošetrovanie duplicitných záznamov

Do lokálnej databáze sa môžu dostať duplicitné záznamy o jednej a tej istej úlohe. K tejto situácii dochádza v prípade, keď sa požadovaný výsledok skladá zo statusu viacerých úloh a obsahuje kompletnú informáciu o ich podúlohách. Za týchto okolností sa do výsledku môže zahrnúť podúloha, ktorá je súčasťou kolekcie úloh a neskôr aj samotná materská kolekcia, v rámci ktorej sa do databáze vložia aj všetky jej podúlohy. Dôjde tak k vytvoreniu duplicitných záznamov pre niektoré úlohy kolekcie. Užívateľ napríklad použije metódu QueryJobs, aby získal všetky svoje bežiace úlohy (stav RUNNING). Ak užívateľ vlastní kolekciu úloh, ktorá je v stave RUNNING, tak sa táto kolekcia zahrnie do výsledku, spolu so všetkými svojimi podúlohami. Veľmi pravdepodobne sa v stave RUNNING nachádzajú aj niektoré podúlohy, preto ich L&B služba tiež zahrnie do výsledku. V niektorých prípadoch totiž môže naopak nastať situácia, keď kolekcia samotná zadaným podmienkam nevyhovuje, ale niektoré jej podúlohy áno, preto sú podúlohy zaraďované do výsledku bez ohľadu na materskú kolekciu. Použitie metódy JobStatus duplicity nevytvára, pretože sa jedná o status jedinej úlohy.

V prípade použitia metódy UserJobs je ošetrenie duplicitných záznamov najjednoduchšie. V gLite prostredí nie je možné meniť vlastníka úlohy, platí tiež, že podúlohy majú vlastníka zhodného s vlastníkom materskej úlohy. Aplikácia tak ukladá iba úlohy, ktoré nemajú žiadneho rodiča. Úlohy, ktoré rodiča majú sa do lokálnej databáze uložia spolu so svojou materskou úlohou.

V prípade QueryJobs je situácia zložitejšia. Rodič nutne nemusí vyhovovať zadanej podmienke, preto horeuvedený postup nie je možné použiť. Aplikácia postupne uloží všetky úlohy a duplicity ošetruje až dodatočne. Po spracovaní výsledku zavolá SQL dotaz, ktorý identifikuje duplicity a tieto záznamy vymaže. Takýto spôsob je z pohľadu databáze efektívnejší ako kontrolovať prítomnosť duplicít priebežne pri ukladaní jednotlivých úloh. Aplikácia vymaže ten duplicitný záznam, ktorý nebol vložený do lokálnej databáze v rámci svojej materskej kolekcie, ale ktorý tam bol uložený ako samostatná úloha. Rozdiel medzi týmito záznamami je v tom, že samostatne uložená úloha nemá v tabuľke children_db_id uloženú informáciu o príslušnosti ku svojej materskej kolekcii (samostatne uložená úloha môže byť ukladaná skôr ako jej rodič, preto databázový identifikátor rodiča v tom čase nemusí byť nutne k dispozícii).

Existuje ešte jeden spôsob, ako môžu v lokálnej databáze vzniknúť duplicitné záznamy. V prípade, že užívateľ zadá viacero dotazov na rovnaký L&B server, v rámci jednotlivých výsledkov sa môžu objaviť tie isté úlohy. Takto vytvorené duplicity aplikácia neošetruje, ich prítomnosť závisí plne na užívateľovi a ním vytvorených endpointoch. V prípade, že by aplikácia už uložené úlohy iba aktualizovala a nie vytvárala nové záznamy, nedalo by sa jednoducho zistiť, ktorá úloha patrí ku ktorému spojeniu. Táto informácia by totiž musela byť uložená v samostatnej tabuľke, pretože jedna úloha by mohla patriť viacerým spojeniam. Pri ukladaní každého statusu by najprv musela prebehnúť kontrola na existenciu záznamu o rovnakej úlohe, čo predstavuje výrazné zvýšenie databázovej záťaže.

Navyše, záznamy reprezentujúce jednotlivé duplicitné úlohy nemusia byť rovnocenné, pretože rôzne dotazy môžu mať nastavené odlišné príznaky určujúce podrobnosti obsiahnuté v stiahnutých statusoch úloh. Duplicity sa preto nedajú ošetriť na úrovni virtuálnych zložiek, neexistuje objektívne kritérium, podľa ktorého by bolo možné určiť konkrétny záznam, ktorý sa má použiť pri zobrazení detailu úlohy a pri prípadnom počítaní štatistík zložky (najaktuálnejší alebo najpodrobnejší záznam, prípadne iné kritérium).

6.2Štatistické údaje o prijatých úlohách

Konštrukcia virtuálnych zložiek určuje istú množinu úloh. GUI umožňuje nad takto definovanými množinami prostredníctvom zásuvných modulov počítať štatistické údaje. Po každej aktualizácii zložky je nutné prepočítať aj štatistiky, aplikácia toto automaticky nezabezpečuje z dôvodu možnej časovej náročnosti tvorby štatistík pri vysokom počte úloh. Konštrukcia

zoznamu úloh patriacich do zložky je nezávislá operácia od počítania štatistík. Opak neplatí, štatistiky je možné počítať iba po načítaní úloh zložky, zásuvný modul postupne prechádza zoznam úloh, načítava potrebné údaje o úlohe, ktoré následne spracováva..

6.2.1Špecifikácia potrebných údajov

Zostavovanie štatistík je potenciálne náročná operácia. Pre každú úlohu zložky je nutné načítať príslušný detail, ktorý sa použije ako podklad pre tvorbu štatistických údajov. Vo väčšine prípadov pluginy nebudú počítať tak komplexné údaje, aby potrebovali plný detail úlohy, ale stačia im len niektoré špecifické parametre. Aplikácia preto rozlišuje 6 kategórii definujúcich, ktoré údaje sa budú načítavať pre potreby zásuvného modulu. Pred vytvorením štatistík aplikácia zistí, do ktorej kategórie spadá vybraný plugin a podľa toho mu dodáva údaje o úlohách.

Jednotlivé kategórie sú označené číslom:

0 – jednohodnotové parametre presne definované vymenovaním

1 – všetky jednohodnotové parametre

2 – ako 1 a navyše všetky viachodnotové parametre, ktoré sa netýkajú podúloh

3 – plný detail úlohy, avšak bez detailu podúloh (iba identifikátory)

4 – plný detail úlohy (spolu s plným detailom všetkých podúloh)

5 – použité jednohodnotové parametre aj viachodnotové parametre sú definované vymenovaním, v prípade, že množina názvov viachodnotových parametrov obsahuje reťazec *childStates*, aplikácia rekurzívne aplikuje túto kategóriu aj na detaily o podúlohách

Definovanie potrebných parametrov pre počítanie štatistík je realizované prostredníctvom kategórií, pretože výskyt parametrov v statuse úlohy sa môže líšiť medzi jednotlivými verziami L&B služby a takýto spôsob zaručuje potrebnú flexibilitu. Kategórie sú navrhnuté tak, aby zoskupovali parametre podľa náročnosti ich použitia. Jednohodnotové parametre sú reprezentované ako stĺpec v tabuľke lb_jobs, viachodnotové je nutné vyhľadávať v ďalších tabuľkách. Najnáročnejšia z pohľadu databázovej práce je postupná konštrukcia detailu aj pre podúlohy. Tvorcovia pluginov by preto mali dôsledne zvážiť, do akej kategórie zaradia vytváraný modul, pretože to bude mať významný vplyv na dĺžku výpočtu štatistických údajov.

6.2.2Rozhranie modulu pre štatistiky

Zásuvný modul pre štatistiky alebo iné potrebné výpočty nad danou množinou úloh musí implementovať rozhranie `StatsPluginInterface`, ktoré zahrňuje nasledujúce metódy:

- init(int folderId, Connection con, Locale locale, String tablePrefix)
- clearStats()
- addJob(JobDetail jd)
- computeAndSaveStats()
- displayStats(JPanel panel)
- getCategory()
- getUsedSimpleParameters()
- getUsedMultivalueParameters ()

Na inicializáciu hodnôt potrebných na spoluprácu pluginu s GUI slúži metóda `init()`. Parameter folderId predstavuje identifikátor zložky, ku ktorej sa dané štatistiky vzťahujú. Ďalej sa pluginu nastaví spojenie na databázu, určí sa použité jazykové nastavenie, aby GUI a pluginy používali rovnaký jazyk a nakoniec sa pluginu predá prefix, ktorým by mal začínať názov tabuľky, do ktorej bude ukladať získané štatistické údaje. Metóda init by mala zabezpečiť aj prípadné načítanie už uložených štatistík z databáze.

Metóda `clearStats()` jednoducho vymaže všetky doterajšie štatistiky, používa sa pred začatím nového výpočtu. V tejto metóde by namalo nedochádzať k zmazaniu uložených štatistík, pri zlyhaní výpočtu by ostali k dispozícii aspoň staršie údaje.

Úloha je zahrnutá do štatistík zavolaním metódy `addJob()`. Plugin môže počítať štatistiky dvoma rôznymi spôsobmi. Prvý spôsob pri každom volaní `addJob()` znovu prepočíta všetky údaje. Táto cesta je vhodná pri jednoduchších štatistikách, keď príslušné výpočty nie sú príliš zložité. Druhá možnosť je zostaviť štatistiky až pri volaní `computeAndSaveStats()`. Údaje obsiahnuté v detaile úlohy, ktorý sa predá tejto metóde ako vstupný parameter závisia na kategórii, do ktorej modul patrí.

Metóda `computeAndSaveStats()` vypočíta štatistiky a uloží ich do databáze. Aplikácia zavolá túto metódu po zahrnutí všetkých úloh do výpočtu prostredníctvom metódy `addJob()`. V prípade, že sa štatistiky počítajú v metóde `addJob()` a zároveň plugin neukladá žiadne údaje do databáze, metóda môže byť prázdna. Zostavovanie štatistík v tejto metóde má výhodu v tom, že sa potrebné výpočty realizujú iba raz. Nevýhodou naopak je vyššia pamäťová náročnosť, pretože plugin si musí pamätať údaje o každej úlohe, ktoré sú potrebné pre daný výpočet, pri priebežnom počítaní štatistík toto nie je nutné. V obecnom prípade sa nedá určiť, ktorý spôsob zostavovania štatistík je vhodnejší, rozhranie preto poskytuje obidve možnosti a výber necháva na tvorcovi pluginu. Táto metóda by zároveň mala zabezpečiť, že staršie údaje o príslušnej množine úloh budú z databáze odstránené.

Na zobrazenie vypočítaných štatistík slúži metóda `displayStats()`, ktorá vykreslí štatistiky na príslušný panel. O presné umiestenie panelu v rámci aplikácie sa ďalej postará GUI, pluginy už na toto nemajú vplyv. Tým sa zaistí, aby nesprávne vytvorený zásuvný modul nenarušil celkový vzhľad aplikácie.

Aplikácia určí kategóriu, do ktorej spadá daný modul prostredníctvom metódy `getCategory()`, ktorá vráti príslušné číselné označenie kategórie.

Ak plugin definuje potrebné parametre vymenovaním (kategórie 0 a 5), aplikácia ich získa zavolaním `getUsedSimpleParameters()`. Metóda vráti množinu textových reťazcov reprezentujúcich názvy vyžadovaných parametrov. Ak modul spadá pod iné kategórie, nepoužíva sa a môže vrátit napríklad hodnotu NULL.

Obdobne funguje metóda `getUsedMultivalueParameters()`, ktorá vráti množinu potrebných viachodnotových parametrov, používa sa pri kategórii 5. Okrem parametrov určených WSDL súborom môže výsledná množina obsahovať hodnotu *children_db_id*, ktorá značí, že sa používajú databázové indexy podúloh (uložené v rovnomennej tabuľke). Reťazec *childrenStates* indikuje, že plugin využíva aj statusy podúloh.

Ak plugin ukladá vypočítané údaje do databáze, mal zaistiť, aby sa názov tabuľky skladal z požadovaného prefixu. Tento prefix sa skladá z konfigurovateľného prefixu spoločného pre všetky tabuľky a z reťazca *plugin_*. V prípade, že užívateľ zmaže virtuálnu zložku, aplikácia postupne prejde všetky tabuľky zásuvných modulov (rozpozná ich podľa názvu, ktorý začína na daný prefix) a vymaže z nich štatistické údaje týkajúce sa odstraňovanej zložky.

Modul môže byť tvorený viacerými triedami v rôznych balíčkoch (package). Všetky triedy predstavujúce implementáciu pluginu musia byť zabalené do jedného súboru JAR[21] (Java Archive), súbor nemusí byť spustiteľný ani nemusí obsahovať triedu Main. Aplikácia však pot-

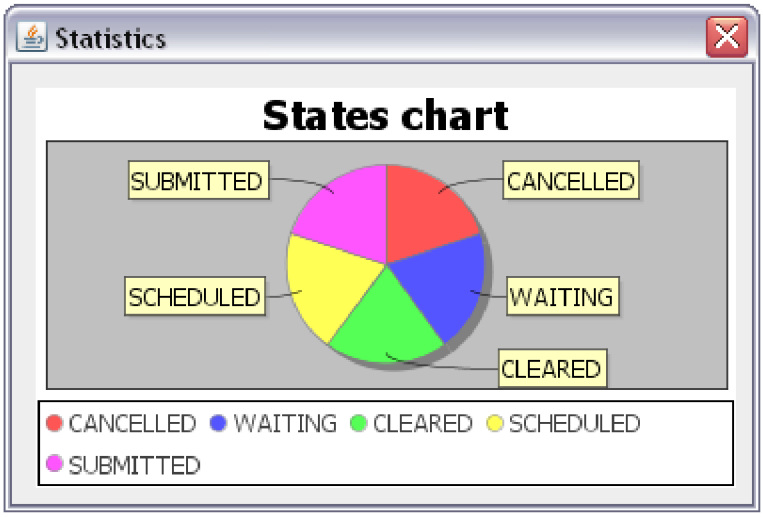
[21] http://java.sun.com/j2se/1.5.0/docs/guide/jar/jar.html

rebuje v rámci JAR súboru identifikovať triedu implementujúcu rozhranie StatsPluginInterface. Z tohto dôvodu je zavedená konvencia, že názov balíčka obsahujúceho túto triedu aj názov triedy samotnej je zhodný s názvom JAR súboru, v názve balíčka sú všetky veľké písmená skonvertované na malé. Aplikácia tak aj v zložitom plugine obsahujúcom viacero tried dokáže identifikovať základnú triedu implementujúcu požadované rozhranie.

6.2.1 Príklady štatistických modulov

V rámci implementácie GUI boli vytvorené štyri zásuvné moduly. Základný plugin sa nazýva `DefaultStatsPlugin`, ak užívateľ nenastaví žiadny zásuvný modul, aplikácia automaticky použije práve tento. Plugin spočíta počet úloh v jednotlivých stavoch a tiež celkový počet úloh v danej množine. Používa iba parameter *state*, preto je zaradený do nultej kategórie. Získané štatistiky jednoducho zobrazí a uloží do lokálnej databáze. Modul je plne internacionalizovaný a lokalizovaný do češtiny a angličtiny. Pri zobrazení použije rovnaký jazyk, aký si užívateľ nastavil v aplikácii.

Zásuvný modul `StatePieChartStatsPlugin` opäť pracuje s informáciami súvisiacimi so stavom úloh, jeho úlohou je demonštrovať schopnosť pluginov prezentovať štatistiky v rôznych podobách. V tomto konkrétnom prípade modul vytvorí z počtu úloh v jednotlivých stavoch koláčový graf.



Obrázok 6.8.: Koláčový graf vykreslený pomocou `StatePieChartStatsPlugin`

Ďalší implementovaný modul sa nazýva `CeNodeStatsPlugin`. Ukazuje príklad štatistík, ktoré sa netýkajú stavu úloh ale iného parametra. Plugin pre každý CE gridu počíta, koľko úloh zo sledovanej množiny sa počas procesu výpočtu spracovávalo na tomto elemente.

Obrázok 6.9.: Štatistiky podľa `CeNodeStatsPlugin`

Jednoduchý príklad pre tvorcov modulov predstavuje `ExemplaryStatsPlugin`. Zostavuje rovnaké štatistiky, ako `DefaultStatsPlugin`, ukazuje však najjednoduchšiu implementáciu rozhrania pre pluginy. Štatistiky sa počítajú priamo v metóde `addJob()`, plugin nie je internacionalizovaný ani lokalizovaný a neukladá vypočítané štatistiky do databáze.

6.2.2Implementácia modulu ExemplaryStatsPlugin

Táto časť bližšie rozoberá implementáciu modulu `ExemplaryStatsPlugin`. Jedná sa o najjednoduchší možný zásuvný modul, ktorý spočíta, koľko úloh sa nachádza v jednotlivých stavoch (viz Príloha B). Na tieto účely modul obsahuje premennú states typu `HashMap<String, Integer>`. Kľúč tvorí názov daného stavu a hodnotou je číslo reprezentujúce príslušný počet úloh. Dalšia existujúca premenná je jobCount, ukladá sa do nej celkový počet úloh danej množiny. V prípade, že užívateľ vyberie tento plugin, GUI zabezpečí vytvorenie inštancie triedy `ExemplaryStatsPlugin` a zavolá metódu `init()`. Keďže plugin neukladá žiadne údaje do lokálnej databáze ani nerešpektuje jazykové nastavenie aplikácie, metóda je prázdna. Pri počítaní štatistík sa ako ďalšia v poradí zavolá metóda `clearStats()`, ktorá nastaví premennej jobCount hodnotu 0 a vytvorí prázdnu mapu states.

Štatistiky sa počítajú priebežne v metóde `addJob(JobDetail jd)`, pretože ich výpočet nepredstavuje žiadnu významnú záťaž. Pri každom zavolaní tejto metódy sa z detailu úlohy vytiahne informácia o stave a v mape states sa zvýši hodnota pre príslušný kľúč. Drobná komplikácia nastáva, ak sa daný stav zatiaľ v mape nenachádza. V tomto prípade sa do mapy pridá nový záznam, kľúč bude názov novopridaného stavu a počet sa inicializuje na 1. Stavy, v ktorých sa žiadna úloha z uvažovanej množiny nenachádza, sa do štatistík vôbec nedostanú. Každé volanie `addJob()` tiež zvýši hodnotu čítača jobCount.

Metóda `getCategory()`vráti číselné označenie kategórie pluginu, v tomto prípade nulu. Metóda `getUsedSimpeParameters()`vráti množinu reťazcov obsahujúcu práve jeden prvok *state*, pretože to je jediný parameter, ktorý je extrahovaný z detailu v metóde

`addJob()`. Navyše existuje rozumný predpoklad, že tento parameter nebude podliehať zmenám pri ďalšom vývoji L&B služby. Metóda `getUsedMultivalueParameters()`vráti NULL.

Obrázok 6.10.: štatistiky vytvorené pomocou `ExemplaryStatsPlugin`

Po načítaní poslednej úlohy GUI zavolá metódu `computeAndSaveStats()`. Plugin však už má štatistiky vypočítané a neukladá ich, preto aj táto metóda je prázdna. Metóda `displayStats()` vykreslí vypočítané štatistiky na príslušný panel, ktorý následne aplikácia zobrazí v novom dialógom okne.

Tento zásuvný modul sa skladá iba z jednej triedy. Aby bola zachovaná zavedená konvencia pre názvy pluginu, výsledný JAR súbor sa volá ExemplaryStatsPlugin.jar a plne kvalifikovaný názov triedy je `exmeplarystatsplugin.ExemplaryStatsPlugin`.

6.3Zobrazenie detailu o úlohe

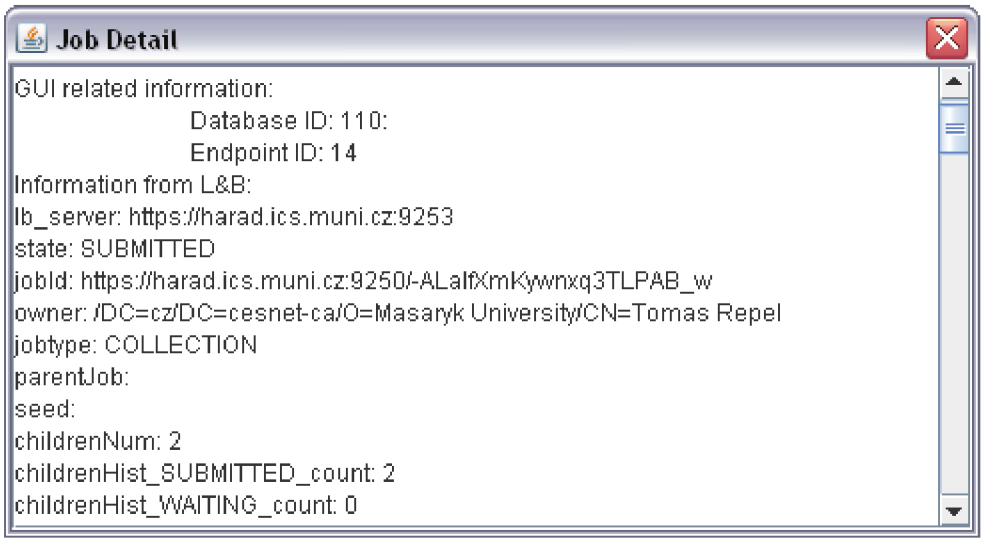
L&B služba poskytuje pomerne komplexnú sumu informácií o úlohe, v takejto zložitej štruktúre nemusí byť jednoduché sa orientovať. V typickom prípade užívateľovi dostačuje iba zlomok informácií obsiahnutých v detaile úlohy. Aplikácia poskytuje možnosť implementovať vlastný zásuvný modul, ktorý požadované informácie dokáže adekvátne zvýrazniť, čím uľahčí užívateľovi orientáciu v jednotlivých parametroch a hodnotách statusu úlohy.

Rozhranie, ktoré musí implementovať každý plugin zobrazujúci detail úlohy sa nazýva `ViewPluginInterface`. Zahrňuje nasledovné metódy:

- displayJobDetail(JPanel panel, JobDetail jobDetail, Locale locale)
- getCategory()
- getUsedSimpleParameters()

•getUsedMultivalueParameters()

Plugin pre detail úlohy tiež poskytuje možnosť špecifikovať, ktoré údaje používa tak, ako to bolo popísané v sekcii o moduloch pre štatistické údaje (viz. 6.9.1). Nová je akurát metóda `displayJobDetail()`. Prvý argument tejto metódy je panel, na ktorom sa daný detail zobrazí. Druhým parametrom je inštancia triedy `JobDetail`, ktorá reprezentuje status úlohy. Posledný argument je objekt triedy `Locale`, ktorý definuje jazykové nastavenie aplikácie.



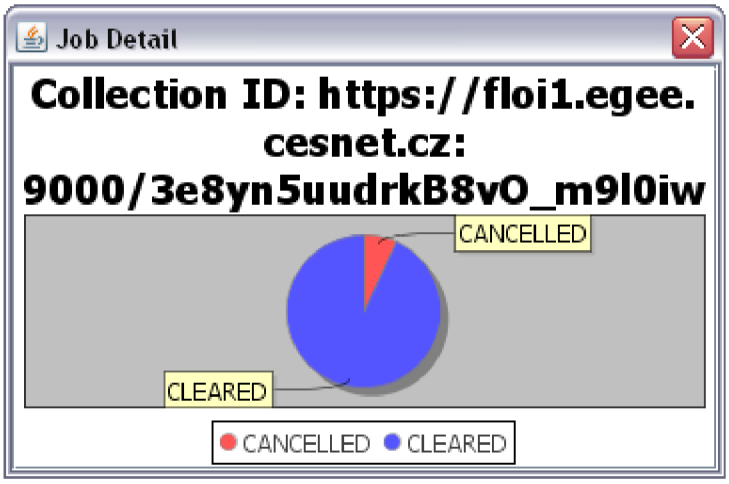
Obrázok 6.11.: Príklad detailu úlohy vytvoreného prednastaveným pluginom

Pri zobrazení a aktualizovaní zložky sa užívateľovi ukáže príslušný zoznam úloh, každá úloha v zozname bude reprezentovaná svojím jobid. Ostatnými údajmi aplikácia v tejto chvíli nedisponuje, pretože udržiavať v pamäti kompletný status pre všetky úlohy zložky by mohlo spôsobiť neúnosne vysoké pamäťové nároky aplikácie. Kompletný stav sa načíta z databáze po kliknutí na konkrétnu úlohu, zavolá sa príslušný zásuvný modul a detail sa zobrazí.

6.3.1Moduly pre zobrazenie detailu úlohy

Ako príklad pre tvorcov zásuvných modulov a tiež ako prednastavený plugin je vytvorený `DefautDetailPlugin`. Tento modul zobrazí hodnoty jednotlivých parametrov vo formáte kľúč: hodnota, kde kľúč je totožný s kľúčom vytvoreným v inštancii triedy `JobDetail`. Ako prvé vypíše parametre, ktoré si k statusu uloží aplikácia pre svoje interné potreby (databázový identifikátor úlohy, id endpointu, ku ktorému daná úloha prislúcha a dátum poslednej aktualizácie úlohy). V prípade viachodnotových parametrov sa jednotlivé hodnoty vypisujú vždy na nový riadok. Kompletný status podúloh sa nezobrazí, vypíše sa iba zoznam jobid týchto úloh.

V gridoch založených na gLite existuje špeciálny typ úlohy zvaný kolekcia. Úloha tohto typu združuje pod sebou niekoľko úloh, samotná kolekcia ale nevyžaduje žiadny výpočet. Pre tento typ úlohy je vytvorený plugin s názvom `CollectionPlugin`, ktorý zobrazí koláčový graf podľa počtu úloh kolekcie nachádzajúcich sa v jednotlivých stavoch. Tento modul slúži na prehľadné zosumarizovanie celkového stavu kolekcie a zároveň demonštruje možnosť užívateľa prispôsobiť si zobrazovanie detailu úlohy svojim individuálnym potrebám



Obrázok 6.12.: Detail kolekcie vytvorený modulom `CollectionPlugin`.

6.3.2Implementácia pluginu DefaultDetailPlugin

Táto časť pojednáva o implementácii modulu `DefaultDetailPlugin`. Dôležitá je primárne implementácia metódy `displayJobDetail()`. V tele metódy sa vytvorí reťazec, do ktorého sa uloží status úlohy, objekt typu `JTextArea`, ktorá daný reťazec zobrazí, a na konci sa inštancia triedy `JTextArea` umiestni na príslušný panel (viz Príloha C).

Zaujímavá je najmä konštrukcia reťazca, do ktorého sa zakóduje príslušný detail úlohy. Na začiatku sa do reťazca vloží názov pluginu. Pomocou get-metód triedy `JobDetail` sa do ďalších dvoch riadkov vloží databázový identifikátor úlohy, identifikátor spojenia na L&B server, ktoré zodpovedá stiahnutie danej úlohy do lokálnej databáze a dátum poslednej aktualizácie úlohy.Nasleduje výpis informácií získaných priamo z L&B servera, pre ktorý sa opäť využijú get-metódy poskytované triedou JobDetail. Implementácia realizujúca pridávanie jednotlivých jednohodnotových parametrov statusu úlohy do výsledného reťazca res vyzerá nasledovne:

```
for (String key : jobDetail.getSimpleProperties().keySet())
  res += key + ": " + jobDetail.getSimpleProperty(key) + "\n";
```

Prvý riadok zabezpečí postupné prechádzanie všetkých kľúčov jednoduchých parametrov, druhý pripojí dvojicu kľúč : hodnota na koniec reťazca. Obdobne sa v tele metódy prechádzajú viachodnotové parametre, užívateľské anotácie a podúlohy danej úlohy.

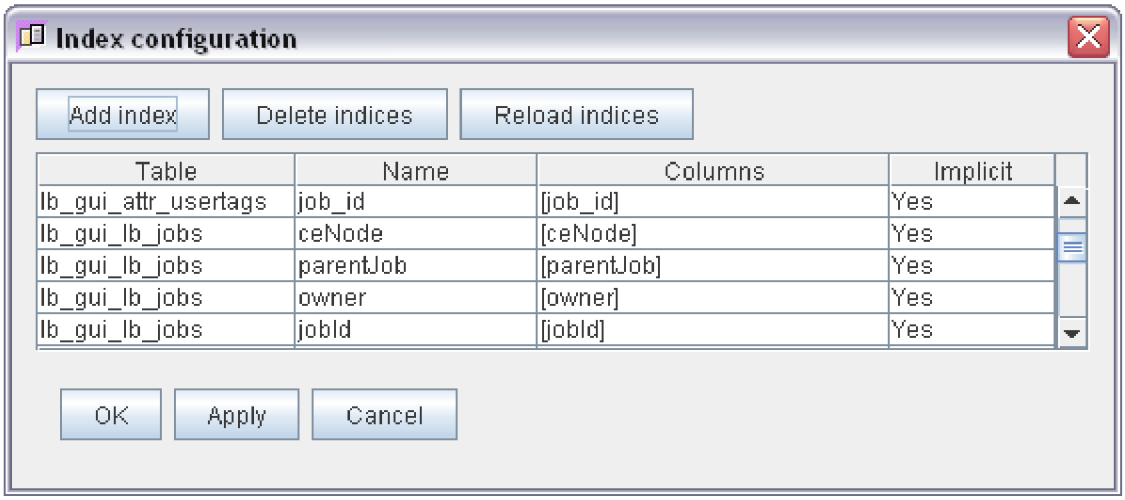
6.4Indexovanie dát v databáze

V gride prebieha množstvo úloh, preto je pravdepodobné, že objem dát stiahnutých do lokálnej databáze bude značný. Pri aktualizácii virtuálnych zložiek dochádza k skenovaniu tabuľky lb_jobs, prípadne ďalších tabuliek obsahujúcich viachodnotové atribúty. Aby vykonávanie databázových operácií bolo čo najefektívnejšie, aplikácia podporuje vytváranie indexov. Rozhranie poskytujúce administráciu indexov je pomerne jednoduché, GUI nemá za úlohu nahradiť stávajúce nástroje na správu databáz. Pri špecifikovaní SQL podmienky v rámci virtuálnej zložky by mal užívateľ skontrolovať, či existuje aspoň jeden index použiteľný pri spracovaní danej podmienky. GUI je schopné pri absencii indexu na parametre vyskytujúce sa v podmienke na túto skutočnosť užívateľa upozorniť.

Aplikácia má definované implicitné indexy, ktorých existenciu overuje pri každom spustení. Ak takýto index neexistuje, tak sa vytvorí. Implicitné indexy vytvorené nad tabuľkou lb_jobs operujú s jedným stĺpcom. Tieto stĺpce sú:

- jobId
- owner
- parentJob
- destination
- ceNode
- lb_request_id
- state

Okrem týchto stĺpcov GUI vytvára aj index operujúci so špeciálnym parametrom stateEnterTimes. V stĺpci stateEnterTimes_DONE_time je uložený čas, v ktorom sa daná úloha dostala do stavu DONE, obdobne existujú stĺpce pre ďalšie možné stavy úloh. Pre každý takýto stĺpec aplikácia vytvára implicitný index. Stĺpce pre implicitné indexy sú vybrané tak, aby urýchľovali očakávané najčastejšie typy dotazov od užívateľov.



Obrázok 6.13.: Administrácia indexov

Ďalšie implicitné indexy sú vytvorené nad tabuľkami ukladajúcimi viachodnotové parametre statusu úlohy. Nad každou takouto tabuľkou je vytvorený index zahrňujúci stĺpec job_id. Typická práca s touto tabuľkou bude zahŕňať nájdenie všetkých hodnôt príslušného parametra pre konkrétnu úlohu, prípadne zistenie, či daná úloha obsahuje v rámci príslušného parametra konkrétnu hodnotu. Oba typy dotazov dokážu efektívne využiť index nad stĺpcom job_id. Rovnaký index sa vytvára aj nad tabuľkou children_db_id.

GUI poskytuje prehľad o existujúcich indexoch nad tabuľkami podieľajúcimi sa na uložení stavu úlohy spolu s jednoduchou administráciou týchto indexov. Pri pridávaní indexu aplikácia užívateľovi poskytne na výber z daných tabuliek, následne ponúkne na výber stĺpce vybranej tabuľky. Užívateľ označí stĺpce ktoré majú byť súčasťou indexu a GUI po potvrdení index vytvorí. Je možné aj mazať indexy, implicitné indexy však nemá zmysel rušiť, aplikácia ich pri opätovnom spustení nanovo vytvorí.

6.4.1Využitie indexov pri urýchlení databázových operácií

Užívateľ môže administrovať indexy priamo prostredníctvom ľubovoľného nástoja na spravovanie databáz, aplikácia však toto poskytuje najmä z dôvodu výrazného urýchlenia databázových operácii za použitia vhodných indexov. Indexy komplikujú vkladanie, mazanie a aktualizáciu údajov, tým dochádza k ďalšiemu spomaleniu práce s endpointami, ktorá už tak je relatívne pomalá z dôvodu volania vzdialených služieb. Ťažisko predpokladanej užívateľskej práce s aplikáciou sa však týka najmä virtuálnych zložiek, kde dochádza k výberu úloh z databáze, preto výhody indexov prevažujú.

Na demonštráciu využitia indexov poslúžia nasledujúce príklady. Máme databázu obsahujúcu 3820 úloh. Index na stav úlohy má pri tomto rozsahu na beh aplikácie minimálny vplyv. Vyhodnotenie dotaz na všetky úlohy v stave DONE, ktorých je približne 10% sa urýchli z priemerných 0,427 s na 0,036 s. To predstavuje zrýchlenie o jeden rád, zložka sa však v oboch prípadoch načíta takmer okamžite. Zaujímavejšia situácia nastane pri počítaní štatistík. Uvažujme zložku obsahujúcu všetky úlohy databáze. Vytvorenie zoznamu úloh zložky trvá v tomto prípade priemerne 9,6 sekundy. Modul `defaultStatsPlugin` spočíta počty úloh v jednotlivých stavoch za necelých 7 sekúnd. Ak však umelo priradíme tomuto pluginu kategóriu 4 (pre každú úlohu sa získa jej kompletný detail vrátane detailu podúloh), počítanie sa predĺži na 103 sekúnd. V prípade, že z databáze odstránime implicitné indexy nad tabuľkami viachodnotových parametrov, dôjde k spomaleniu vyhľadávania týchto parametrov pre daný detail úlohy. Počítanie spomínaných štatistík následne potrvá až 134 sekúnd.

Nevýhody indexov sa ukážu pri vytváraní a obnovovaní spojení na L&B server. Pred uložením prípadne aktualizovaním úloh aplikácia zhodí stávajúce indexy a po skončení tieto indexy znova vytvorí. Tento postup je vo väčšine prípadov rýchlejší od postupného aktualizovania indexov ako sú do tabuliek vkladané ďalšie a ďalšie záznamy. Zhodenie aj nahodenie indexu nad tabuľkou s 3820 úlohami trvá približne 2 sekundy. Uvažujme spojenie, ktoré vo výsledku obsahuje iba šesť úloh. Ich uloženie do prázdnej databáze zabralo priemerne 11,3 sekundy. Práca s indexami predĺžila tento čas až na 85 sekúnd. Časový rozdiel zhruba zodpovedá množstvu vytváraných implicitných indexov.

6.5Konzument webovej služby L&B servera

Na spodnej úrovni aplikácie prebieha komunikácia medzi L&B serverom a GUI. Triedy zabezpečujúce túto komunikáciu sú realizované pomocou zásuvného modulu. Tento modul neslúži na prispôsobenie GUI individuálnym potrebám užívateľov, ako tomu bolo u doteraz spomínaných pluginov, motiváciou pre realizovanie konzumenta vo forme zásuvného modulu bolo zachovať čo najväčšiu nezávislosť aplikácie na zmenách WSDL popisu služby. Implementácia konzumenta webovej služby často závisí priamo na tomto popise. Existujú automatizované nástroje, napríklad spomínaný AXIS, ktoré dokážu z WSDL vytvoriť triedy jazyka Java reprezentujúce zástupný kód volania vzdialenej služby. Pri každej zmene WSDL by bolo nutné tieto triedy nanovo vygenerovať a prekompilovať celú aplikáciu. Konzument realizovaný ako zásuvný modul zaručuje, že aplikačná vrstva je úplne nezávislá na konkrétnej implementácii komunikácie s L&B službou. Je možné implementovať viacero pluginov, aplikácia dovoľuje nastaviť konkrétny modul, ktorý bude používať.

6.5.1Rozhranie pre konzumenta L&B služby

Rozhranie, ktoré musí konzument implementovať sa nazýva `LBConsumer`. Zahrňuje štyri metódy:

- setConfigMap(Hashtable<String, String> configMap)
- setEndpoint(Endpoint endpoint)
- callLBMethod()
- saveResult(Object result, String attrPrefix, Connection con)

Metóda `setConfigMap()` predá konzumentovi iniciálne nastavenie získané zo súboru `config.ini`. Toto nastavenie je reprezentované objektom typu Hashtable<String, String>, ktorý uchováva mapovanie kľúča na príslušnú hodnotu.

Všetky potrebné údaje na spojenie a získanie dát z L&B servera obaľuje trieda `Endpoint`, metóda `setEndpoint()` nastaví konzumentovi daný objekt.

Metóda `callLBMethod()` použije príslušné údaje, realizuje volanie vzdialenej metódy a vráti výsledok. Návratový typ je obecný typ `Object`.

Získaný výsledok sa predá metóde `saveResult()` ako jej prvý vstupný argument. Druhý argument je prefix pre názvy tabuliek s viachodnotovými parametrami a tretí je spojenie na lokálnu databázu. Metóda výsledok spracuje a uloží ho do lokálnej databáze.

Konzumenti pre jednotlivé metódy klientského API sa mierne líšia. Aplikácia pri vytváraní konzumenta (triedy implementujúcej rozhranie `LBConsumer`) musí špecifikovať konkrétnu metódu, ktorú bude daný konzument vzdialene volať. Balíček do ktorého patrí `LBConsumer` preto obsahuje aj rozhranie `LBConsumerFactory`. Toto rozhranie má iba jedinú metódu nazvanú `getConsumer(String methodKey)`. Reťazec methodKey identifikuje požadovanú vzdialenú metódu. Môže nadobúdať nasledovné hodnoty:

- user_jobs
- query_jobs
- job_status

Úplná implementácia pluginu tak pozostáva z tried predstavujúcich konzumentov jednotlivých metód a z triedy implementujúcej rozhranie `LBConsumerFactory`, ktorá podľa zadaného kľúča metódy vráti príslušného konzumenta.

Uloženie do databáze predstavuje moment, keď sa dáta z L&B servera transformujú do podoby, s ktorou GUI dokáže pracovať. Priamo so získanými dátami pracuje iba konzument, ktorý presne pozná ich vnútornú štruktúru. Pri implementácii konzumenta musí tvorca pluginu z inštancie triedy Endpoint vytiahnuť vstupné údaje zadané užívateľom a preformátovať ich do podoby vhodnej pre vzdialené volanie konkrétnej metódy. Trieda `Endpoint` obsahuje na tento účel príslušné get-metódy. Následne musí konzument realizovať volanie samotné. Oba tieto kroky nie sú implementačne náročné. O niečo zložitejšia sa javí byť akurát implementácia metódy `saveResult()`.

6.5.1 Metóda saveResult

Celkový status úloh má istú vnútornú štruktúru, preformátovanie a následné uloženie týchto dát do lokálnej databáze môže byť pomerne netriviálna úloha. Tvorca pluginu musí poznať presnú schému tabuliek, prípadne WSDL popis a spôsob, ako z neho aplikácia danú schému vytvára. Tiež musí správne implementovať spracovanie špeciálnych parametrov (viz 6.5.2) a naplnenie tabuľky children_db_id (viz 6.7). Ďalej metóda musí k informáciam získaným z L&B servera uložiť aj dáta pre internú potrebu aplikácie. Tieto dáta sa ukladajú priamo do tabuľky lb_jobs do nasledovných stĺpcov:

- id – databázový identifikátor úlohy, lb_jobs používa tento stĺpec ako primárny index. Je nastavený ako auto-increment, databáza hodnotu pre tento stĺpec vygeneruje automaticky

- lb_server – adresa L&B servera, z ktorého sú dáta o danej úlohe

- lb_request_id – obsahuje databázový identifikátor endpointu, v rámci ktorého sa daná úloha stiahla do lokálnej databáze

- last_update – uchováva dátum, kedy došlo k poslednej aktualizácii danej úlohy v lokálnej databáze

Aplikácia umožňuje užívateľovi pre každé spojenie rozhodnúť, či sa pri obnovení endpointu úlohy kompletne vymažú a znova uložia alebo sa iba aktualizujú jednotlivé záznamy novými údajmi. Prvá možnosť je vhodná, ak užívateľ potrebuje získať aktuálnu informáciu a nezaujímajú ho staršie výsledky. Ak si napríklad potrebuje prezrieť všetky svoje aktuálne bežiace úlohy, úlohy z predošlého volania sa zmažú a stiahnu sa nové. Aktualizácia úloh je vhodná, ak užívateľ nechce stratiť úlohy, ktorých priebeh skončil pomerne dávno a L&B služba už

o danej úlohe neeviduje žiadne údaje. Pri aktualizácii sa záznamy o úlohách, ktoré vo výsledku neboli nemenia. V prípade preuloženia sa o zmazanie starých údajov postará aplikácia samotná, metóda zabezpečí uloženie nových dát. V prípade aktualizácie musí metóda záznam preuložiť. Samozrejme ak sa informácia o danej úlohe v lokálnej databáze nenachádza, je potrebné ju vložiť ako nový záznam.

6.5.1Existujúce implementácie konzumenta

V rámci tejto práce boli vytvorené dva zásuvné moduly konzumentov. Prvý je implementovaný na základe API vytvorenom v rámci bakalárskej práce *Java API pro dotazovací rozhraní služby EGEE LB* [14]. Jadro tohto API tvoria triedy vygenerované z WSDL popisu nástrojom AXIS, ktoré sú následne využité v ďalších triedach zjednodušujúcich užívateľovi volanie L&B služby, prácu s certifikátmi a ustanovením šifrovaného spojenia. Ak užívateľ nešpecifikuje inak, aplikácia použije tento modul. Ukázalo sa, že konzumenti implementujúci jednotlivé metódy sa líšia len v spracovaní vstupných argumentov a v zavolaní príslušnej metódy. Obdržaný výsledok z L&B servera predstavuje vo všetkých prípadoch zoznam statusov úloh, okrem metódy JobStatus, kde výsledok reprezentuje len jeden status konkrétnej úlohy. Na ten sa dá pozerať ako na jednoprvkový zoznam, preto aj implementácia metódy `saveResult()` je rovnaká pre všetkých konzumentov. S výhodou je tak možné použiť koncept abstraktnej triedy, v ktorej bude implementovaná spoločná funkcionalita. Kompletný modul tvorí 5 tried, prvá implementuje `LBConsumerFactory`, druhá je abstraktná trieda nazvaná `DefaultConsumer` a implementuje `LBConsumer` rozhranie. Ďalšie tri triedy predstavujú konzumentov pre jednotlivé metódy a sú rozšírením triedy `DefaultConsumer`. Využitie abstraktnej triedy tieto triedy výrazne zjednoduší, napríklad konzument pre bezparametrickú metódu UserJobs obsahuje iba jednu metódu nazvanú `call()`, ktorá zavolá príslušnú vzdialenú metódu a vráti výsledok.

Zdrojový kód metódy `call()` vyzerá nasledovne:

```
protected Object call() throws Exception {
  return sc.userJobStates(); }
```

kde `sc` reprezentuje príslušný zástupný kód vzdialenej služby, tzv. stub. Podrobnejšie informácie o použití tohto API nájdete v [14].

Druhý zásuvný modul je tiež implementovaný pomocou knižnice AXIS, ale nepoužíva žiadne vygenerované triedy. Pracuje na úrovni SOAP protokolu. AXIS poskytuje nástroje pre zostavenie SOAP požiadavky na vzdialené volanie služby. Túto správu následne pošle na príslušný server a obdrží odpoveď, opäť ako obecnú SOAP správu. Na spracovanie získanej správy AXIS ponúka výkonný XML parser. Práca na úrovni SOAP nie je taká pohodlná ako využitie API vyššej úrovne, na druhej strane nie je nutné obsah SOAP správy transformovať na objekty a naopak, preto v obecnom prípade je volanie vzdialenej služby pomocou tohto kon-

zumenta menej časovo náročné. Obdobne ako pri implementácii prvého pluginu aj tu bol využitý koncept abstraktných tried. Implementácia protokolu SSL nie je rovnaká v prípade Javy a komponent využívajúcich niektoré verzie Globus Toolkit, preto tento konzument obsahuje triedy, ktoré dovoľujú použiť iba SSLv3 bez šifrovania RC4, inak by mohlo dôjsť k zlyhaniu ustanovenia šifrovaného spojenia.

Kapitola 7

Záver

Úlohou tejto práce bolo navrhnúť grafické užívateľské rozhranie schopné komunikovať s L&B službou prostredníctvom klientského API využívajúceho webové služby. Vytvorená aplikácia toto umožňuje, dokáže extrahovať dáta z rôznych L&B serverov, ktoré následne uloží do lokálnej databáze. GUI na komunikáciu s L&B službou používa zásuvné moduly, preto nie je priamo závislé na poskytovanom API a do istej miery sa tak dokáže prispôsobiť možným budúcim zmenám tohto API analyzovaním WSDL popisu využívaných webových služieb.

Po uložení úloh do lokálnej databáze je možné pracovať v off-line režime. Užívateľ získa plnú silu dotazovacieho jazyka SQL. V tomto jazyku zadáva podmienky, ktoré špecifikujú množinu vyhovujúcich úloh a nad takto determinovanými množinami GUI dovoľuje počítať rôzne štatistické údaje. Tieto výpočty sú opäť realizované ako zásuvný modul, prostredníctvom implementácie vlastného pluginu užívateľ dokáže získať akékoľvek štatistické údaje, ktoré potrebuje v jemu vyhovujúcom grafickom prevedení.

Aplikácia dovoľuje sústrediť pozornosť nielen na štatistické údaje ale aj na jednotlivé úlohy, užívateľovi sa sprístupní celkový status úlohy ako ho odvodila L&B služba. Grafické prevedenie tejto informácie opäť závisí na zvolenom zásuvnom module, vytvorenie vlastného pluginu umožňuje prispôsobiť výzor detailu úlohy užívateľovým špecifickým potrebám. Využitie jazyka Java robí aplikáciu prenositeľnú na rôzne platformy.

Budúci vývoj bude pravdepodobne pozostávať z implementácie nových zásuvných modulov podľa potrieb jednotlivých užívateľov, prípadne z vytvorenia ďalších konzumentov využívajúcich iné nástroje pre pracovanie s webovými službami. Komplexnejší rozvoj bude vyžadovať rozšírenie funkcionality o prácu s jednotlivými udalosťami. Je však možné, že táto práca bude vyžadovať príliš rozdielny prístup a ako vhodnejšie sa ukáže byť vytvorenie nezávislej aplikácie. Ďalší vývoj tiež môže súvisieť s budúcim rozšírením funkcionality klientského API, napríklad so zvažovaným zavedením podpory pre postupné streamovanie objemných výsledkov, čím by sa stalo zbytočné rozdeľovať dotaz na viacero menších dotazov (čo do množstva úloh obsiahnutých vo výsledku) podľa časového razítka.

Literatúra

[1] Foster, I., Kesselman, C.: *The Grid2: Blueprint for a New Computing Infrastructure(The Elsevier Series in Grid Computing)*, Morgan Kaufmann Publihers, 2003, 1-55860-933-4. 2.1

[2] Foster, I.: *What is the Grid? A Three Point Checklist*, 2002, http://www.mcs.anl.gov/~itf/Articles/WhatIsTheGrid.pdf (november 2009) 2.2

[3] Sarmenta, L. F.G.: *Bayanihan: Web-Based Volunteer Computing Using Java*, Lecture Notes in Computer Science, Vol. 1368, Springer-Verlag, 1998. http://groups.csail.mit.edu/cag/bayanihan/papers/wwca98/html/ (november 2009). 2.3

[4] Kmuníček, J.: *Grids as key phenomenon of information technology of new millennium*, ICS MU Bulletin, ISSN 1212-0901, vol. 16, no. 2: 1–5, 2005, http://ics.muni.cz/~kmunicek/gridy.pdf (november 2009). 2.3

[5] Foster, I., Kesselman, C., Tuecke S.: *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*, International J. Supercomputer Applications, http://www.globus.org/alliance/publications/papers/anatomy.pdf (november 2009). 2.4

[6] Kuba, M.:*Gridové služby*,http://www.ics.muni.cz/~makub/soap/grid.html (november 2009).

[7] EGEE-II, http://egee.cesnet.cz/cs/info/About_EGEE_CZ.pdf (november 2009). 3.1.2

[8] Laure, E. a ďalší: *Programming the Grid with gLite*, Computational Methods in Science and Technology, 9/2006. 3.2

[9] Foster, I.: *Globus Toolkit Version 4: Software for Service-Oriented Systems*, In IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, pp 2-13, 2006. 3.2

[10] Laure, E. a ďalší: *Middleware for the next generation Grid infrastructure*, Computing in High Energy Physics and Nuclear Physics (CHEP04), 2004. 3.2

[11] Burke, S., Campana, S., Lanciotti, E., Lorenzo, P. M., Miccio V., Nater, C., Santinelli, R., Sciabà, A.: *gLite 3.1 User Guide*, http://edms.cern.ch/file/722398/1.2/gLite-3-UserGuide.pdf (november 2009). 3.2

[12] Matyska, L., Křenek, A., Ruda, M.,Sitera, J., Kouřil, D., Voců, M., Pospíšil, J.,Mulač, M., Salvet, Z.: *Job Tracking on a Grid - the Logging and Bookkeeping and Job Provenance Services*, CESNET technical report number 9/2007. 3.3, 4

[13] Kuba, M.: *Web Services*, DATAKON 2006.http://www.ics.muni.cz/zpravodaj/articles/269.html (november 2009). 5, 5.2

[14] Kramec, T.: *Java API pro dotazovací rozhraní služby EGEE LB*, bakalárska práca FI MU, 2009. 5, 6.12.3

Príloha A

Príklad vstupného súboru config.ini

```
//url to mysql database
mysql_db = jdbc:mysql://localhost/lb_db
//credentials for mysql database, if key db_password is not
found, applicaton asks for it after being launched
db_login = lb_db
db_password = lb_db
//path to log file
log_path = ./lb_gui_client.log
//path to certificate for client authentization, format PEM
user_cert = ./myCert.pem
//password for client certificate, if private key is
password //protected, empty value represents empty password,
application //asks for password, if key is missing
user_cert_password =
//path to file with key (PEM format)
user_cert_key = ./myKey.pem
//path to trusted CA files
ca_files = ./grid-security/certificates
//path to WSDL for LB
wsdl_path = ./LB.wsdl
//path to directory containing plugin (jar file with directory
with required libraries)
plugin_dir = ./plugins
//path to directory containing plugins for stats
stats_plugin_dir = ./statsPlugins
//SSL socket timeout
timeout = 600000
//prefix for table names
table_prefix = lb_gui_
```

Príloha B

Zásuvný modul ExemplaryStatsPlugin

```
public class ExemplaryStatsPlugin implements StatsPluginInterface
{
  private HashMap<String, Integer> states;
  private int jobCount;

  public boolean clearStats() {
    states = new HashMap<String, Integer>();
    jobCount = 0; }

  public boolean addJob(JobDetail jd) {
    String state = jd.getSimpleProperty("state");
    if (states.containsKey(state))
      states.put(state, new Integer(states.get(state)+1));
    else states.put(state, new Integer(1));
      jobCount++; }

    public boolean computeAndSaveStats() {}

  public boolean displayStats(JPanel panel) {}

  public void init (int mainId, boolean isVirtFolder, Connection
con, Locale locale, String tablePrefix) {}

  public Set<String> getUsedSimpleParameters() {
    Set<String> usedParams = new HashSet<String>();
    usedParams.add("state");
    return usedParams;
  }

  public Set<String> getUsedMultivalueParameters() {
    return null;
  }

  public int getColumnCategory() {
    return 0;
  }
}
```

Príloha C

Zásuvný modul DefaultDetailPlugin

Vybrané časti zdrojového kódu metódy `displayJobDetail()`

```
displayJobDetail(JPanel pane, JobDetail jobDetail, Locale loc) {
  //insert name of plugin to job detail
  String res = resources.getString("plugin_used");

              .  .  .

  //insert simple properties to job detail
  for (String key : jobDetail.getSimpleProperties().keySet()) {
    res += key + ": " + jobDetail.getSimpleProperty(key) + "\n";}

  //insert multivalue properties to job detail
  for(String key: jobDetail.getMultivalueProperties().keySet()) {
    res += key + ":\n";
    for (String val : jobDetail.getMultiValProperty(key)) {
      res += "\t" + val + "\n";
    }
    res += "\n";
  }

  //insert userTags to job detail
  for(String key :jobDetail.getUserTags().keySet()) {
    res += "\t" + key + ": ";
    for(String val: userTags.get(key)) {res += "" + val + ", ";}
    res = res.substring(0, res.length()-2) + "\n";
  }

              .  .  .

}//end of displayJobDetail()
```

Príloha D

K práci je priložené CD s nasledovným obsahom:

- Zdrojové kódy GUI a súvisiacich knižníc a zásuvných modulov
- Spustiteľný súbor
- Text práce vo formáte ODT a PDF