Katedra Informatiky
Fakulta Matematiky, Fyziky a Informatiky
Univerzita Komenského, Bratislava

Grid Computing
Implementácia služby v Globus Toolkite

(Diplomová práca)

Bc. Peter Bajči

Školiteľ: RNDr. Andrej Bebják

Bratislava, 2010

Prehlásenie

Čestne prehlasujem, že som túto diplomovú prácu vypracoval samostatne s použitím citovaných zdrojov.

. . . . . .

Poďakovanie

Ďakujem RNDr. Andrejovi Bebjákovi za cenné rady pri písani diplomovej práce.

Abstrakt

BAJČI, PETER: *Grid computing - implementácia služby v Globus toolkite.* (Diplomová práca) - Univerzita Komenského v Bratislave, Fakulta matematiky, fyziky a informatiky - vedúci: RNDr. Andrej Bebják. Bratislava : UK, 2010, 85 strán.

Cieľom tejto práce je poskytúť čitateľovi prehľad v oblasti grid computingu, predstaviť a charakterizovať techniky použité v Globus toolkite na stavbu grid aplikácii a implementovať grid službu pomocou nástrojov Globus toolkitu. V prvej kapitole sa venujeme základným princípom grid computingu, jeho jednotlivým oblastiam a v krátkosti rozoberáme smery, ktorými sa vývoj uberá. V druhej časti sa sústreďujeme na Globus toolkit a popisujeme jeho architektúru i infraštruktúru, na základe ktorých je implementovaný. V závere kapitoly priblížime jednotlivé komponenty, ktoré toolkit tvoria a popisujeme ich funkcionalitu. V záverečnej kapitole práce sa venujeme vývoju modelovej grid aplikácie pre menšiu lokálnu sieť pomocou nástrojov toolkitu. Postupne prechádzame jednotlivými krokmi vývoja od prípravy prostredia, cez dizajn aplikácie až po samotnú implementáciu služby a vysvetľujeme špecifické postupy implementácie v Globus toolkite.

Kľúčové slová: grid, grid computing, globus toolkit, webové služby, implementácia, grid služba, GRAM.

Abstract

BAJČI, PETER: *Grid Computing - Service Implementation in Globus Toolkit.* (Master Thesis) - Comenius University in Bratislava, Faculty of Mathematics, Physics and Informatics - advisor: RNDr. Andrej Bebják. Bratislava : UK, 2010, 85 pages.

The aim of this work is to provide the reader with an overview of grid computing, introduce and characterize techniques used in Globus Toolkit to build grid applications and implement a grid service with the tools of Globus toolkit. In the first chapter we focus on the basic principles of grid computing, its areas and we discus the research objectives shortly. In the second part we focus on the Globus Toolkit itself and describe both, the architecture and infrustructure, on basis of which it is implemented. At the end of the chapter we list the individual components of the Globus Toolkit and describe their functionalities. In the last chapter of the thesis we devote our attention to development of a model grid application for a small local network using the Globus toolkit tools. We follow the individual steps of the implementation beginning with the environment setup, through application design, to service implementation itself and describe the specific implementation procedures of the Toolkit.

Keywords: grid, grid computing, globus toolkit, web services, implementation, grid service, GRAM.

Obsah

Úvod

Rastúci výkon pracovných staníc v rôznych organizáciách spôsobuje, že často len zlomok ich hrubého výkonu je využitý na pokrytie nárokov aplikácií samostatného užívateľa. Zároveň však rastú nároky a potreby celej organizácie na vykonávanie náročných výpočtových úloh či už vedeckého, alebo biznis charakteru. Zozbieranie prebytočného výkonu staníc sa preto stáva jasnou a efektívnou alternatívou finančne náročných superpočítačov. Túto agregáciu umožňujú súčasné sieťové technológie, ktoré zabezpečujú zdanlivo jednoduché a rýchle prepojenie existujúcich pracovných staníc a ich komunikáciu.

Zozbieranie a využitie prebytočných výpočtových zdrojov za pomoci sieťových technológií je jednou z hlavných úloh Grid computingu. Poprednú rolu vo výskume a hlavne vývoji hrá Globus Alliance, ktorá poskytuje rozsiahly súbor nástrojov a aplikácií pod názvom Globus Toolkit, ktoré riešia úlohy, ktorým Grid computing čelí.

V posledných rokoch sa vo veľkých organizáciách Grid technológie úspešne uplatňujú. Jedným príkladom za všetky môže byť výpočtový Grid slúžiaci potrebám centra CERN pre analýzu výstupných dát urýchľovača LHC. Táto práca si však dáva za úlohu preskúmať možnosti aplikácie technológií Grid computingu v menších organizáciách. Globus Toolkit preukázal, že je výborne škálovateľný na veľké projekty, no my sa budeme zaoberať jeho špecifikami v malých a stredných lokálnych sieťach. Hlavným cieľom tejto práce je pomocou Globus Toolkitu implementovať službu pre takúto sieť. V práci ukážeme

špecifické programovacie postupy typické pre Globus Toolkit a zhodnotíme jeho použiteľnosť v takýchto scenároch.

Kapitola 1

Grid computing

Vízia a zameranie Grid computingu sa mení. Ešte v minulom desaťročí Ian Foster a Carl Kesselman navrhli definíciu, ktorá popisovala Grid ako „hardvérovú a sofvérovú infraštruktúru, ktorá poskytuje spoľahlivý, konzistentný, trvácny a lacný prístup k najvýkonnejším výpočtovým kapacitám“[1]. Táto oblasť je ešte stále nová a pohľad na ňu sa stihol za posledné roky zmeniť. V súčasnosti sa zameriava viac na zdieľanie zdrojov a spoluprácu ako na zhlukovanie veľkej výpočtovej sily do superpočítačov. V tejto novej pozícii môžeme Grid definovať ako “typ paralelného a distribuovaného systému, ktorý umožňuje zdieľanie, výber a agregáciu geograficky distribuovaných autonómnych zdrojov za behu, v závislosti od ich dostupnosti, schopností, výkonu, ceny a užívateľovych požiadaviek na kvalitu služieb.“[2]

Rozdiely medzi distribuovanými výpočtami a Grid computingom môžu byť na prvý pohľad pomerne nejasné. Požiadavky kladené na Grid systém sú však oveľa komplexnejšie ako požiadavky na distribuovaný systém. Distribuované počítanie zjednocuje potenciálne stovky až tisíce homogénnych systémov za účelom využitia ich spoločnej výpočtovej sily, ktorá by ináč

[1] FOSTER, I. - KESSELMAN, C. 1998. *Computational Grids*. 1998.

[2] FOSTER, I. 2002. *What is the Grid? A Three Point Checklist* In *GridToday*. Júl, 2002.

bola izolovaná. Naproti tomu, cieľom Grid computingu je efektívne využitie heterogénnych systémov v neobmedzenej škále. Dôraz je na využitie všetkých možných zdrojov organizácie, ktoré koordinovane potom fungujú ako väčšie jednotky výpočtových zdrojov. Medzi tieto zdroje môžeme zaradiť procesorový čas, úložný priestor, servery, siete, periférne zariadenia a všetky informácie. Zameranie je na schopnosť podporovať takéto zjednocovanie a zdieľanie zdrojov naprieč mnohými administratívnymi doménami.

Kľúčovým pojmom v dosahovaní takto vytýčených cieľov je virtuálna organizácia[3]. Dynamické zdieľanie, ktoré je v centre našej pozornosti, je o prístupe k vzdialeným počítačom, softvéru, dátam a úložným priestorom v rámci ľubovoľnej úlohy. Keďže toto zdieľanie využíva zdroje potenciálne veľkého počtu nezávislých organizácií, bude podliehať rozsiahlemu riadeniu v podaní poskytovateľov aj užívateľov zdrojov. Tí budú chcieť explicitne ovládať, čo, kedy, za akých podmienok a kto môže zdieľať. Množina účastníkov, zdrojov a inštitúcií definovaných takýmito vzťahmi, ktorí sa podieľajú na spoločnej úlohe, tvorí virtuálnu organizáciu.

Príkladom virtuálnej organizácie môže byť skupina špecialistov z geograficky rozptýlených výskumných centier participujúcich na výskumnom fyzikálnom projekte, výpočtový výkon počítačov z týchto centier určený na beh simulácií k projektu a diskové kapacity, na ktoré sa ukladajú výsledky takýchto simulácii. Zdroje jedného centra sa môžu zapájať do viacerých projektov zároveň a vytvárať tak ďalšie virtuálne organizácie poskytovaním zdrojov pre tieto projekty.

Vlastníci zdrojov pritom kladú podmienky, kedy a ktoré zdroje budú sprístupnené pre jednotlivé virtuálne organizácie. Preto sa vzťahy v nich dynamicky menia v čase. Virtuálne organizácie umožňujú nezávislým skupinám, organizáciám a jednotlivcom zdieľať prostriedky kontrolovateľným

[3]FOSTER, I. - KESSELMAN, C. - TUECKE, S. 2001. *The Anatomy of the Grid:* Enabling Scalable Virtual Organizations. 2001. s. 3-5.

spôsobom. Ich členovia tak môžu spolupracovať za účelom dosiahnutia spoločného cieľa. Vytváranie, manažment a následné využívanie práve takýchto dynamických virtuálnych organizácií je predmetom Grid computingu.

1.1 Prebiehajúce projekty v oblasti Grid computingu

Vo svete sa rapídne zvyšuje počet projektov využívajúcich Grid technológie. Patria sem vedecké projekty z oblasti prírodných vied (predovšetkým fyzika, chémia, informatika a astronómia), medicíny, klimatológie a ekológie. Rovnako existuje rastúci počet súkromných spoločností, ktoré sa rozhodli pomocou technológií Gridu účelne využiť technologickú infraštruktúru svojej organizácie na dosahovanie organizačných cieľov. V rámci Grid computingu sa vyčlenilo niekoľko odvetví, každé so svojimi špecifikami, či už v architektúre systémov, alebo v podmienkach prostredia, v ktorých aplikácie z oblasti pracujú.

1.1.1 Scientific Grids

Enabling Grids for E-sciencE (EGEE) je európskym vedeckým projektom financovaným priamo zo zdrojov Európskej únie a je v súčasnosti najväčšou multidisciplinárnou Grid infraštruktúrou. V súčasnosti už zhlukuje spolu viac ako 250 centier zo 48 krajín[4] a vytvára tak škálovateľnú sieť výpočtových zdrojov, ktorá slúži európskej a svetovej vedeckej komunite. Agregované zdroje v súčte obsahujú viac ako 40 000 procesorov, poskytujú 5PB diskového priestoru a výkonajú viac ako 100 000 pracovných jednotiek denne. V rámci EGEE Gridu existuje už viac ako 150 virtuálnych organizácii, ktoré sa

[4]MATYSKA, L. 2008. *Empowering Grids – the EGEE gLite middleware* 2008.

venujú výskumu z oblasti informatiky, astrofyziky, vysokoenergetickej fyziky a fúzie, výpočtovej chémie a ďalších prírodných vied.

Infraštruktúra[5] tohto projektu je založená prevažne na otvorenom sprostredkovateľskom softvéri gLite, pričom v menšinovej podobe sú používané aj komponenty Globusu. Kooperácia týchto komponentov je zaručená vďaka štandardizovaným rozhraniam OGSA. Projekt sa postupne vyvinul z Európskeho DataGrid programu a momentálne sa nachádza vo svojej tretej, stabilnej fáze. Medzi inými spracováva aj dáta z LHC. Svojim charakterom podobnými Gridmi sú TeraGrid a Open Science Grid.

1.1.2 Public resource computing

Svojím charakterom sa tento prístup trochu vymyká presnej definícií Gridu. Nejde totiž o zjednocovanie zdrojov viacerých organizácii, ale o zjednocovanie výpočtového výkonu jednotlivých používateľov[6]. Tí sa do Gridu pripájajú na báze dobrovoľnosti a poskytujú tak výkon svojich osobných počítačov pre účely rôznych, zväčša výskumných projektov. Fakt, že sa môže prihlásiť ľubovoľný dobrovoľník a anonymita internetu, spôsobujú unikátne bezpečnostné výzvy. Tvorca virtuálnej organizácie má minimálnu možnosť ovládať jednotlivé zdroje a má tak pomerne pasívnu pozíciu. Musí tiež rátať s potenciálne škodlivým správaním užívateľa a zamedziť takýmto aktivitám. Medzi používané techniky patrí duplicitné počítanie tej istej úlohy a aktívna prevencia zahltenia. Ďalším problémom je častá lokálna nedostupnosť jednotlivých užívateľov a z toho prameniace vysoké percento nedokončených úloh.

Jednoznačne vedúcim projektom v tejto oblasti je Berkeley Open Infrastructure for Network Computing (BOINC). Samotný BOINC je sprostredkovateľský softvér pracujúci na princípe klient/server. Na strane servera umož-

[5] EGEE Technical Infrastructure [online]. `http://technical.eu-egee.org/index.php?id=134`

[6] ANDERSON, D.P. 2004. *BOINC:* A System for Public-Resource Computing and Storage. In /em 5th IEEE/ACM International Workshop on Grid Computing. 2004.

ňuje vedeckým tímom založiť vlastný projekt, vytvárať a posielať úlohy dobrovoľníkom a zbierať výsledky výpočtov. Základom serverovej strany je relačná databáza, v ktorej sú uložené informácie o pracovných jednotkách, výsledkoch, dobrovoľníkoch a aplikáciách. Funkcie sú potom výkonávané sadou webových služieb. Plánovací server manažuje spojenie s užívateľom, priraďuje úlohy participantom a preberá naspäť výsledky. Dátový server zbiera výsledkové súbory pomocou mechanizmu založeného na certifikátoch pre zaistenie bezpečnosti. Sťahovanie súborov z pohľadu participanta je založené na jednoduchom FTP. Pre dobrovoľníkov je k dispozícii sada klientov pre operačné systémy Windows, Unix a MacOS. Klientská aplikácia sa spája so serverom a spúšťa samotné výpočtové úlohy.

V agregovanej výpočtovej sile projekt BOINC vykazuje zaujímavé čísla. Priemerne je v každom momente pripojených približne 600 000 počítačov s výkonom okolo 1800 TeraFLOPov [7]. Svojím zameraním, konceptom, aj rozsahom je podobný projekt World Community Grid, ktorého tvorcom a sponzorom je firma IBM.

1.1.3 Enterprise computing

Motivácia pre použitie Grid technológií v komerčnej sfére je podobná, ako pri vedeckých projektoch. Ide o nájdenie, združenie a využitie technologických zdrojov, ktorými spoločnosť disponuje. Keďže takáto spoločnosť sa pohybuje v konkurenčnom prostredí, musí sa svoje zdroje snažiť využívať čo najefektívnejšie. Nevyužitý výkon osobných počítačov a pracovných staníc dáva priestor Grid technológiám na efektívnu utilizáciu týchto zdrojov, pričom sa šetria prostriedky za nákup drahého hardvéru. Spoločnosti, pre ktoré sú správne informácie základným zdrojom konkurenčnej výhody, môžu ťažiť z

[7] ANDERSON, D.P. - REED, K. 2008. *Celebrating Diversity in Volunteer Computing.* 2008.

tohto prístupu.

Vhodnými aplikačnými oblasťami môžu byť farmaceutické spoločnosti vyvíjajúce nové lieky, inžinierske firmy testujúce konštrukcie, alebo investičné spoločnosti analyzujúce riziká portfólia. Samozrejme, so zmenou aplikačnej oblasti sa čiastočne menia aj požiadavky na systém. V prvom rade, zdieľanie zdrojov sa zameraním presúva skôr do vnútra organizácie, ako na vonok, medzi viaceré organizácie. Otázka bezpečnosti a autorizácie používateľov je ešte dôležitejšia, keďže systém bude pravdepodobne obsahovať informácie kritické pre biznis a ich strata by ohrozila konkurenčnú výhodu spoločnosti, a tým pádom aj jej existenciu. Posledným dôležitým bodom je nutnosť čiastočnej centralizácie správy systému, čo je vo výskumných Gridoch vyslovene nežiaduce. Niekoľko globálnych spoločností už poskytuje riešenia založené na Grid technológiách pre organizácie. Patria medzi ne napríklad IBM, Oracle, alebo Hewlett-Packard. Riešenia pre túto sféru sa často opierajú o open source softvér, akým je Globus Toolkit a prispievajú tak k interoperabilite systémov v rámci štandardu OGSA. Rozšírenosť operačného systému Windows v obchodnej sfére však často určuje platformu, na ktorej je systém nasadený. Zaujímavým projektom je Alchemi [8], ktorý relatívne jednoducho umožňuje koštrukciu Gridu na sieti s OS Windows, vývoj aplikácií pre túto sieť a zároveň zachováva schopnosť spolupracovať s webovými službami iného sprostredkovateľského softvéru pre Grid. Alchemi je implementovaný na platforme Microsoft .NET.

1.1.4 Utility computing

Utility computing stelesňuje víziu poskytovania výpočtových prostriedkov ako bežnú službu pre verejnosť, podobne ako je to s poskytovateľmi pitnej

[8]LUTHER, A. - BUYA, R. - RANJAN, R. - VENUGOPAL, S. *Alchemi:* A .NET-based Enterprise Grid Computing System. 2006.

vody, elektriny, alebo plynu. Tento prístup dáva obrovskú výhodu pri využívaní IT v podobe nulových nákladov na zavedenie, keďže výpočtové zdroje sú prenajímané od poskytovateľov.

Vhodnými výpočtovými jednotkami pre realizáciu Utility computingu sú clustre. Tie však postrádajú niektoré dôležité charakteristiky, ktoré sú pre takúto službu nutné, ako sú flexibilita, škálovateľnosť a (relatívne) jednoduché užívateľské rozhranie. Preto sa zmenila paradigma a zaviedli sa Grid technológie založené na webových službách. V takejto architektúre clustre tvoria základnú výpočtovú jednotku Gridu ovládanú vlastným systémom manažmentu zdrojov. Grid systém potom pôsobí o úroveň vyššie a neriadi samotné zdroje, ale iba spolupracuje so systémami manažmentu zdrojov jednotlivých clustrov. Tieto musia byť na druhej strane upravené o rozhranie, ktoré takúto spoluprácu umožňuje.

Špecifiká týchto systémov výplývajú z ich podstaty platenej služby. V prvom rade, poskytovatelia výpočtových služieb musia do svojich systémov zahrnúť infraštruktúru pre umožnenie obchodovania s touto komoditou. Táto infraštruktúra typicky pozostáva z:

- Adresár Grid služieb: pomocou neho zoznamuje poskytovateľ svojich zákazníkov s dostupnými typmi služieb.

- Obchodný server: komunikuje so zákazníkovým Grid klientom a na základe oceňovacích algoritmov predáva prístup ku zdrojom podľa dohodnutej ceny a požadovaného typu služby.

- Oceňovacie algoritmy: určujú optimalizované ceny, za ktoré budú zákazníkový poskytované zdroje v závislosti od požadovanej obchodnej stratégie poskytovateľa, súčasného zaťaženia systému a kvality požadovanej služby zákazníkom.

- Účtovanie: slúži na zaznamenávanie využitia zdrojov a účtuje užívate-

ľom poplatky podľa dohody medzi obchodným serverom a klientom.

Keďže rôzni zákazníci nepravidelne požadujú rôznu kvalitu, intenzitu a objem služieb, je veľmi ťažké predpovedať dostupnosť určitých zdrojov. Preto sa implementujú rôzne stratégie včasnej rezervácie. V neposlednom rade je dôležitá bezpečnosť, ktorá sa zameriava na ochranu zverených dát. V tejto oblasti operuje niekoľko globálnych spoločností, ale začínajú sa objavovať aj menší hráči so svojimi riešeniami. Známe sú služby ako Amazon EC2, Sun Grid Compute Utility, alebo GoGrid. Každá sa pritom vyznačuje určitými odlišnosťami.

Počas posledných rokov je vývoj najmä v tejto oblasti rýchly. Spolu s evolučnými zmenami v prístupe k poskytovaniu výpočtovej sily pre zákazníkov sa zmenil aj termín, toto odvetvie je v súčasnosti označované ako Cloud computing. Tento termín nezahrňuje len poskytovanie Grid infraštruktúry pre výpočty na vyžiadanie (označované ako Infrastructure as a Service, IaaS), čo je stelesnením Utility computingu, ale aj poskytovanie softvérových služieb (Software as a Service, SaaS) a poskytovanie platforiem pre vývoj a spúšťanie aplikácii (Platform as a Service, PaaS). Napriek tomuto vývoju Cloud computing stále stavia na technológiách Gridu a webových službách.

1.2 Výskum a smerovanie Grid computingu

1.2.1 Štandardizácia a integrácia

Výrazným súčasným trendom a zároveň nutnosťou je štandardizácia. Grid systémy sú často zložené z mnohých kooperujúcich nástrojov, ktoré spolu vytvárajú celkovú funkcionalitu. Štandardizované rozhrania týchto nástrojov sú jediným možným riešením efektívnej komunikácie nielen služieb v rámci Gridu, ale aj komunikácie medzi rôznymi Gridmi. OGSA (Open Grid Services Architecture) poskytuje práve toľko potrebnú štandardizáciu rozhraní

pre rôzne komponenty Gridu. V súčasnosti sú vypracované a popísané rozhrania k službám, ktoré sú potrebné pre beh každého Gridu. Samozrejme, vývoj OGSA stále pokračuje, priebežne sa pracuje na ďalších rozhraniach a je isté, že s ďalším výskumom v oblasti prídu ďalšie esenciálne komponenty pre Grid, ktoré bude potrebné štandardizovať. V súčasnosti existuje niekoľko projektov, ktoré sa zaoberajú vývojom sprostredkovateľského softvéru určeného na stavbu Grid systémov. Trendom ostáva čoraz presnejšie dodržiavanie štandardu OGSA, čo zvyšuje interoperabilitu systémov založených na odlišných technológiách. Medzi popredné projekty v oblasti patria nasledujúce:

- Globus Toolkit sa stal de facto standardom Grid computingu. Je to implementácia architektúry definovanej v OGSA a WSRF špecifikácie, pričom veľký počet Grid riešení sa zakladá práve na jeho komponentoch. Kľúčovým stavebným kameňom sú stavové webové služby.

- Condor nie je klasickým sprostredkovateľským softvérom pre Grid sám o sebe, ale umožňuje zbieranie nevyužitých CPU cyklov na počítačoch, na ktorých je nasadený a ich efektívne koordinované využitie. Spojením s GT v podobe Condor-G poskytuje vhodné riešenie pre vzdialené výpočty na Gride.

- GLite je relatívne jednoduchý sprostredkovateľský softvér vyvinutý pre projekt EGEE, ktorý na ňom stavia. Hlavným cieľom bola pri jeho tvorbe robustnosť a stabilita. Používa súčasti z iných projektov orientovaných na Grid.

- Alchemi je vytvorený špeciálne pre operačné systémy Windows a implementovaný na základe technológie Microsoft .NET. To robí tento sprostredkovateľský softvér zajímavým v prvom rade pre biznis sféru. Komunikácia so štandardizovanými komponentmi je tiež zaručená.

- UNICORE, na rozdiel od sady komponentov, ponúka vertikálne integrované riešenie, ktoré sa sústredí na jednotný prístup k distribuovaným zdrojom. Taktiež je v súlade so štandardmi ako je OGSA a nie je tak ťažké dosiahnuť spoluprácu s GT. Je vyvíjaný konzorciom nemeckých univerzít.

- Legion predstavoval trochu odlišný prístup ku Grid softvéru. Bol postavený na jednotnom objektovom modeli a jednotnom rozhraní medzi službami. Projekt skončil svoj vývoj v roku 2001, ale ostáva naďalej dostupný.

1.2.2 Podpora CASE nástrojov

Rýchlosť vývoja aplikácií pre Grid je stále relatívne pomalá. Súčasné programovacie jazyky a vývojové prostredia vo všeobecnosti nepodporujú tvorbu Grid aplikácií na takej úrovni, aká by bola potrebná. Dôvodom je skutočnosť, že jazyky, ktoré sa za bežných okolností považujú za jazyky vyššej úrovne, ako napríklad java, C++, alebo python, zohrávajú v Gride rolu nižšieho stupňa. Ten si totiž berie abstrakcie z týchto úrovní ako virtuálne stroje a ďalej ich rozširuje a spája.

Pre programátorov Grid aplikácií by bolo oveľa jednoduchšie pracovať s celým systémom ako jedným superpočítačom, pričom vývojové prostredie by vytváralo abstrakciu nad jednotlivými virtuálnymi strojmi. Manažment virtuálnych organizácií, pod ktorý spadajú činnosti ako prieskum zdrojov, komunikácia medzi službami, alebo delegácia autorizovaných identít, by bol v čo najväčšej možnej miere skrytý pred programátorom, ktorý by sa tak mohol sústrediť na funkčnú časť softvéru a nie na automatizovateľnú réžiu.

Prekážkou jednoduchej realizácie takéhoto prístupu je samotný spôsob vývoja Grid nástrojov. Rôzne komponenty, ktoré slúžia na stavbu Grid systémov, sú vyvíjané nezávisle. Kooperujú vďaka dodržiavaniu štandardov, no

tento proces štandardizácie ešte stále nie je ukončený a preto je ťažké zaručiť bezchybnú spoluprácu týchto komponentov, ak s nimi má pracovať programátor iba sprostredkovane, cez vývojové prostredie. V neposlednom rade je otázkou aj výrazná zložitosť a rozmanitosť operácií na decentralizovanom Gride. Kvôli tejto komplexnosti je značne problematické zaručenie efektívnej práce Gridu, ak bude réžia ponechaná na vývojové prostredie.

1.2.3 Sémantické Grid systémy

Web a Grid majú viacero spoločných menovateľov. Preto sa črtá aj spoločné riešenie niektorých nastupujúcich problémov. Jedným z nich je aj prudko narastajúci objem dát a z toho rezultujúce zahltenie informáciami a problematickosť vyhľadávania a spracovania. Riešením pre Web je realizácia vízie sémantického webu, ktorého hlavnou agendou je vybudovanie aparátu, ktorý okrem prezerania dát ľuďmi, umožní aj automatizované spracovanie informácií počítačmi. Užitočnosť podobných mechanizmov v Gride je pre vedecké komunity zjavná a aplikácia technologií sémantického webu v Gride sa stáva dôležitým pre jeho evolúciu. Grid v súčasnosti stojí na služobne orientovanej architektúre a prístup založený na agentoch sa zdá byť vhodným spôsobom, ako zabezpečiť autonómne entity schopné spolupracovať a riešiť úlohy v Gride.

Vízia sémantického Gridu[9] je dosiahnuť vysoký stupeň ľahko použiteľnej automatizácie pre zabezpečenie flexibilnej spolupráce a výpočtov v globálnom meradle, pomocou počítačovo spracovateľných znalostí v Gride. Okrem klasických požiadaviek na architektúru Gridu (ktorým sa budeme podrobnejšie venovať neskôr) pribúdajú ďalšie. Abilita autonómneho správania sa odkazuje na schopnosť systému samokonfigurácie vzhľadom k potrebám množstva

[9]DE ROURE, D. - JENNINGS, N.R. - SHADBOLT, N.R. 2005. *The Semantic Grid: Past, Present, and Future*. In *Proceedings of the IEEE*. vol. 93, no. 3. 2005. s.2

užívateľov pri meniacich sa podmienkach, a takisto aj schopnosť aktívneho zotavovania sa z chýb. Kontextovo senzitívny Grid a kontextovo založené rozhodovanie v ňom by malo podávať užívateľom informácie v správnom čase, správnom formáte a v správnej miere detailu. Táto schopnosť výrazne napomáha v práci s informáciami rozsiahlych Gridov. Dôležitou požiadavkou je aj integrácia dát. Systém by mal byť schopný zmysluplných dopytov do odlišných dátových systémov, alebo inak povedané, požadujeme vyspelú interoperabilitu dát. Toto je jednou z hlavných úloh technógií sémantického webu.

Pre splnenie týchto požiadaviek je potrebné zapojiť viacero technológií z rôznych oblastí informatiky. Okrem bežných webových služieb, ktoré sa stali de facto štandardnou súčasťou Grid architektúry, sa zapájajú aj nasledovné prvky[10]:

- Ontológie a odvodzovanie: ontológie vytvárajú pomocou matematickej logiky definície a vzťahy medzi dátami. Vytvárajú sa rozšírením pôvodného obsahu o metadáta a efektívne tak konceptualizujú celý popisovaný obsah. Predmetom takejto konceptualizácie môže byť čokoľvek: objekty, zdroje, schopnosti, vzťahy, rozhrania, atď. Na vzniknutú ontológiu s vytvorenými konceptami a vzťahmi nadväzujú rôzne odvodzovacie systémy, ktoré pracujú na základe odvodzovacích pravidiel matematickej logiky. Štandardom pre web, ktorý je využiteľný aj v prostredí Gridu, je OWL (Web Ontology Language). OWL je odporúčaním konzorcia W3C.

- Sémantické webové služby sú evolúciou webových služieb, pre ktoré technológie ako WSDL a SOAP neposkytujú dostatočnú mieru abstrakcie, ktorá by umožnila v potrebnej miere automatizáciu procesov

[10]DE ROURE, D. - JENNINGS, N.R. - SHADBOLT, N.R. 2005. *The Semantic Grid: Past, Present, and Future.* In *Proceedings of the IEEE.* vol. 93, no. 3. 2005. s.4-6

ako je prieskum, konfigurácia, spolupráca a spúšťanie služieb. Na takýto účel slúži OWL-S (Web Ontology Language for Services), ktorý je prirodzeným rozšírením OWL a obsahuje sémantické popisy služieb v deskripčnej logike. Ďalšou inciatívou je SWSI (Semantic Web Service Initiative), ktorá na popis služieb používa jazyk logiky prvého rádu.

- Softvéroví agenti: výskum v oblasti multiagentových systémov má mnohé nepriame aplikácie pre sémantický Grid. Agenti totiž podporujú dynamické rozhodovanie, decentralizáciu, koordináciu a autonómne správanie potrebné pre realizáciu virtuálnych organizácii. Je možné vidieť paralelu medzi agentmi a služobne-orientovaným modelom, ak agenti preberajú rolu producentov, sprostredkovateľov a konzumentov služieb. V multiagentových systémoch agenti tvoria koordinované spolupracujúce tímy, podobne ako v Gride interagujúce služby vytvárajú virtuálne organizácie. Niektoré koncepty môžu byť preto v budúcnosti prospešné pri implementácii Grid aplikácií s vysokou požadovanou mierou flexibility.

Kapitola 2

Globus Toolkit

Globus Toolkit je spravovaný a zároveň aj kontinuálne vyvíjaný medzinárodnou spoločnosťou vystupujúcou pod menom Globus Alliance. Samotná aliancia je komunitou organizácií a súkromných osôb, ktoré sa podieľajú na vývoji technológií, všeobecne označovaných ako „Grid“ a je aktívnym členom v oblasti vývojárov takto orientovaného softvéru. Aliancia pôsobí ako partner pri vývoji dôležitých projektov v oblasti vedy a obchodu, pri ktorých je zdieľanie dát a výpočtového výkonu kľúčové.

Globus Toolkit je sada softvérových nástrojov, ktoré je možné využiť pri tvorbe Grid aplikácií a systémov. Pôvodne bol zameraný takmer výlučne na vedecké projekty, no v posledných rokoch je možné badať zvýšené používanie Grid technológií aj v oblasti komerčných aplikácii a Globus Toolkit sa tomuto trendu prispôsobil. Rieši úlohy, ktoré sa typicky vyskytujú pri stavbe zdieľaných systémov, akými sú napríklad prístup k dátam na vzdialenom počítači, bezpečnostná politika, a iné. Jeho vývoj a distribúcia je založená na open source licencii. Otvorená stratégia vývoja bola zvolená po vzore Linuxu, keďže umožňuje širší prístup kvalifikovaných špecialistov z oblasti k danému projektu a následne aj rýchlejší priebeh technických inovácií. Prvá verzia Globus Toolkitu vznikla v roku 1998, momentálne sú aktuálne štvrté

a piate vydania.

2.1 Základné koncepty

Globus Toolkit 4 je na rozdiel od predchádzajúcej verzie vystavaný na čiastočne odlišnej sade špecifikácií. Pôvodnými boli OGSA (Open Grid Services Architecture) a OGSI (Open Grid Services Infrastructure), ktoré vydalo Global Grid Forum. OGSI bolo založené na definícii Grid služby a špecifikovalo spôsob ako klienti interagujú s takouto službou. Určovalo, ako sa služba volá, jej dátové rozhranie, bezpečnostné rozhranie a iné. Dvomi základnými stavebnými kameňmi pre GT4 sú OGSA a WSRF (Web Services Resource Framework). Tieto špecifikácie definujú celú jeho štruktúru, pričom GT4 je ich implementáciou. OGSA vychádza z predchádzajúcej verzie, pričom boli pridané iba najnutnejšie časti z OGSI, ktoré bolo vynechané úplne. Na jeho miesto prichádza WSRF, ktoré prinieslo zmenu podstaty Grid služby.

2.1.1 Open Grid Services Architecture - OGSA

Každý Grid systém je založený na princípe virtuálnych organizácií. Jeho cieľom je integrovať, virtualizovať a spravovať zdroje a služby v takýchto distribuovaných, heterogénnych, dynamických organizáciách. Aby bolo možné tento cieľ dosiahnuť, je potrebné prekonať prekážky, ktoré znemožňujú prístup k rôznym počítačom, dátam, službám a iným zdrojom z rôznych systémov. Tento prístup musí byť dosiahnutý nezávisle od fyzickej lokality. Na architektúru Grid systému sú tak kladené požiadavky a OGSA je architektúrou, ktorá ašpiruje na ich splnenie. Podľa správy [1] organizácie Global Grid Forum sú tými požiadavkami nasledovné:

[1] GLOBAL GRID FORUM. 2005. *The Open Grid Services Architecture, Version 1.0.* 2005.

1. interoperabilita a podpora heterogénnych, dynamických prostredí: rozsah Grid systému určuje aj to, že musí pracovať na viacerých platformách, fungovať v rámci rôznych operačných systémov, pracovať s rôznymi externými zariadeniami a prepájať väčší počet virtuálnych organizácií. Väčšinou je takýto systém zamýšľaný ako dlhodobý a je vysoko pravdepodobné, že sa hardvérové a softvérové vybavenie postupom času bude meniť, takžie riešenie musí byť čo najviac nezávislé na počiatočných podmienkach;

2. zdieľanie zdrojov viacerých organizácií: jednou z hlavných úloh Grid systému je umožniť využívanie zdrojov z nesúvisiacich domén, či už v rámci jednej, alebo vo viacerých organizáciách. Preto musí architektúra umožňovať vytvorenie takýchto mechanizmov. Tie zároveň implikujú isté bezpečnostné požiadavky, ktoré sú jedným z ďalších bodov;

3. optimalizácia: okrem toho, že zdieľanie zdrojov musí fungovať medzi rôznymi organizáciami, podmienkou je, že ich priraďovanie bude fungovať aj efektívne. Optimalizácia prebieha na dvoch stranách. Na strane ponuky zdrojov vo forme umožnenia dynamickej alokácie zdrojov pomocou rôznych stratégií, a na strane dopytu po zdrojoch meraním a monitorovaním pracovného zaťaženia, dopytu po zdrojoch a následným rozhodovaním o prípadnom prerozdelení zdrojov;

4. zabezpečenie kvality služieb (QoS): dôležité služby, týkajúce sa exekúcie úloh a manažmentu dát, by mali posktyovať vopred dohodnutý štandard kvality. Tento štandard musí byť merateľný, a preto je potrebné zaviesť určité metriky, na ktorých základe bude možné túto kvalitu služieb posudzovať;

5. vykonávanie úloh: vyjadruje manažment spracovania úlohy od jej zaregistrovania, cez samotné spustenie, odovzdanie výsledkov práce, až

po zvládnutie výnimiek a chýb. Tento manažment pritom musí plynulo pracovať s množstvom heterogénnych zdrojov;

6. dátové služby: v rôznych vedeckých, ale aj biznis aplikáciách je potrebné pracovať s veľkými obnosmi dát, ktoré sú v prípade Grid systému distribuované. V špeciálnych podmienkach Gridu je potrebné premyslieť spôsob prístupu k dátam, uniformné vyhľadávanie v heterogénnych prostrediach, transport veľkých obnosov dát a tiež udržiavanie konzistentnosti dát, keďže sa očakáva vznik viacerých replík tých istých dát vo veľkých systémoch;

7. bezpečnosť: kľúčovou otázkou pri realizácii Grid systému je práve bezpečnosť všekých služieb. Prístup k službám musí byť zaistený pomocou robustných, bezpečných protokolov a podľa zvolenej bezpečnostnej stratégie. Zaisťovanie prístupu k službám vyžaduje autentifikáciu užívateľa na overenie jeho identity a následne autorizáciu, aby služba bola použitá spôsobom súhlasiacim s bezpečnostnou stratégiou. Bezpečnostné zložky Grid systému musia taktiež byť v súlade s bezpečnostnými systémami hostiteľských prostredí;

8. zníženie nákladov na údržbu: zložitosť komplexných systémov, akými sú aj Grid systémy, zvyšuje pravdepodobnosť výskytu rôznych druhov chýb. Podpora administrácie systému pomocou zavádzania automatizácie znižuje výskyt ľudských chýb a efektívne tak prispieva k znižovaniu nákladov na údržbu;

9. škálovateľnosť: s rastom, alebo vývojom organizácie musí rásť aj jej systém. Vzhľadom na túto povahu je teda zrejmé, že jeho architektúra musí byť schopná sa vysporiadaťŤ a efektívne manažovať využívanie obrovského počtu zdrojov, za dodržania podmienky ďalšej rozšíriteľnosti. Pri stále sa zväčšujúcom portfóliu zdrojov je pritom nutné udržať

výborný výpočtový výkon;

10. dostupnosť služieb: keďže sa počíta s nasadením Grid technológií aj do kritických oblastí, veľká pozornosť musí byť venovaná aj dostupnosti služieb, ktoré poskytuje. Spoľahlivosť záleží od schopnosti prevencie chýb, cez zotavovanie až po diagnostiku a automatickú adaptáciu;

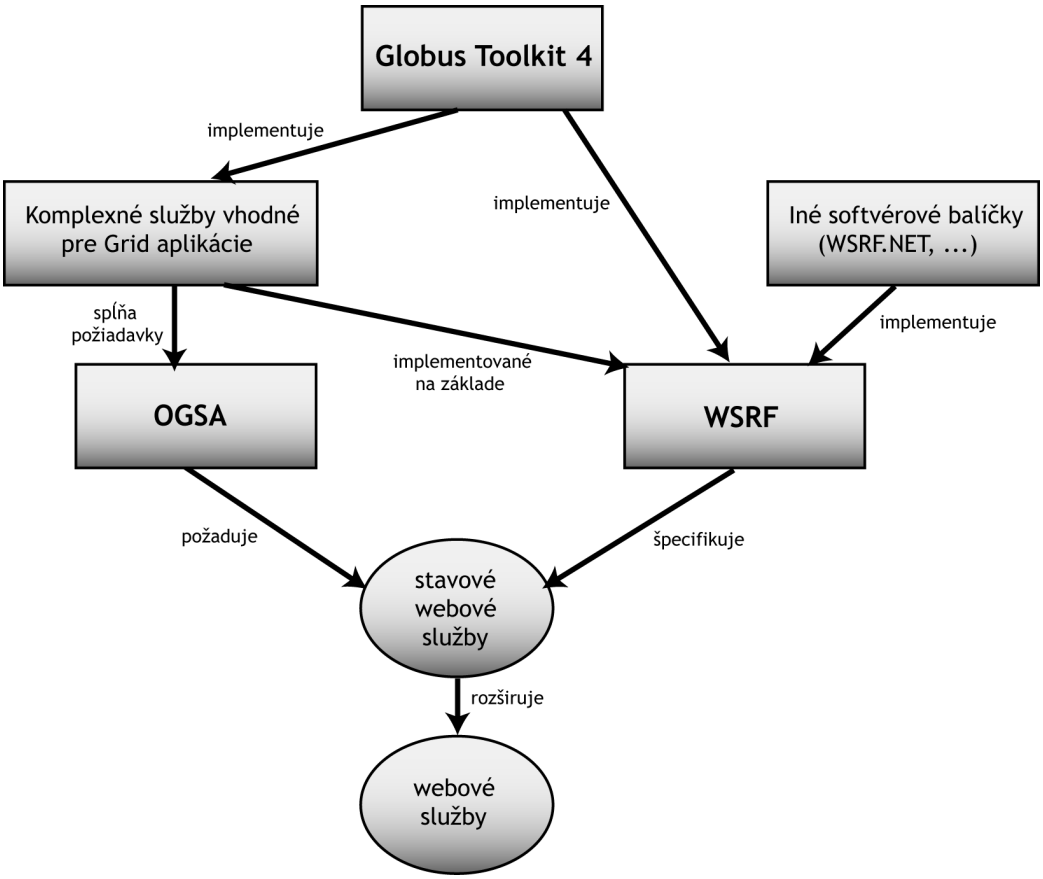
11. jednoduchosť použitia a rozšíriteľnosť: prostredie musí užívateľovi pomocou rozhrania poskytovať dostatočnú mieru abstrakcie, aby bolo možné relatívne jednoducho systém obsluhovať. Na druhej strane, pokročilí užívatelia a administrátori potrebujú operovať na nižších úrovniach, a preto by mali mať možnosť si stupeň abstrakcie riadenia zvoliť. Nedá sa predpovedať, aké komponenty budú Grid aplikácie v budúcnosti potrebovať a akú funkcionalitu poskytovať, a preto je posledná požiadavka, ktorú kladieme na OGSA - jednoduchý vývoj a spustenie nových komponentov, či služieb.

Riešením tejto vízie je štandardizácia. OGSA sa snaží definovať jednotné rozhrania pre všetky typické služby, ktoré Grid systém bežne poskytuje. Táto štandardizácia rozhraní je kľúčová pri vytváraní komponentov schopných vzájomnej komunikácie, prenositeľnosti a znovu použiteľnosti. Tým, že určuje rozhrania služieb, určuje presne aj ich správanie a požadovanú funkcionalitu, s ktorou môžu používatelia, alebo rôzne komponenty pracovať.

2.1.2 Web Services Resource Framework - WSRF

OGSA, ako už z názvu vyplýva, však špecifikuje iba architektúru Grid systému. Chýba však jeho spoločná infraštruktúra. Tou musí byť nejaký vhodný typ sprostredkovateľského softvéru, ktorý zaručí štandardné volanie operácií proklamovaných v rozhraniach. Zároveň však musí táto infraštruktúra umožňovať splnenie podmienok, ktoré sú kladené na OGSA. Ako sme už niekoľ-

kokrát spomenuli, OGSA pracuje so službami. Presnejšie, architektúra požaduje stavové (stateful) služby[2]. Na výber bolo k dispozícií viacero existujúcich sprostredkovateľských riešení, no nakoniec komunita zvolila inú alternatívu.



Obr. 2.1: Vzťah medzi OGSA, WSRF a GT4

Pre účely Gridu sa javili ako vhodné webové služby, avšak tento koncept bol ešte rozšírený a modifikovaný tak, aby v plnej miere splňoval podmienku stavovosti. Spoločnou iniciatívou Globus Alliance a komunity, ktorá stojí za

[2]SOTOMAYOR, B. - CHILDERS, L. 2006. *Globus Toolkit 4*: Programming Java Services. San Francisco : Morgan Kaufmann Publishers, 2006. ISBN-10 0-12-369404-3. s.15

webovými službami, tak vznikol WSRF (Web Services Resource Framework). V mnohom nadväzuje na predchádzajúcu infraštruktúru OGSI a je vlastne jej prepracovaním s využitím pokrokov dosiahnutých pri vývoji webových služieb.

Globus Toolkit vo verzii 4 je postavený na týchto predstavených špecifikáciách. V prvom rade obsahuje komplexné služby, ktoré môžu byť priamo využité na stavbu Grid aplikácií. Týmito službami sú napríklad bežné bezpečnostné a dátové služby, systém riadenia výpočtov, alebo služba pre monitorovanie zdrojov Gridu. Tieto služby spĺňajú väčšinu požiadaviek, ktoré predstavuje OGSA. GT4 teda nie je dokonalou implementáciou OGSA, ale formou jej čiastočnej realizácie. Väčšina spomenutých služieb je postavených na základe WSRF, ktorého kompletnú implementáciu GT4 obsahuje. WSRF je teda základným stavebným kameňom, z ktorého GT4 ťaží. Definuje Stavové webové služby, ktoré architektúra OGSA vyžaduje.

2.2 Webové služby

Ako sme už spomenuli, existuje viacero technológií, ktoré by mohli spĺňať požiadavky na vytvorenie distribuovaného systému. Mohli by sme menovať technológie ako Enterprise Java Beans, RMI, alebo CORBA. Prečo sú ale webové služby najvhodnejším riešením? V prvom rade sa javia ako ideálne pre vytváranie voľne previazaných systémov. Tým pádom sú vhodné pre Grid aplikácie, pretože pri nich sa počíta s veľkou rôznorodosťou v rámci virtuálnych organizácií a ich voľným prepojením pomocou intranetu organizácie, ale aj globálneho internetu. Sú dva hlavné dôvody, ktoré ich robia na tento účel vhodné.

1. Webové služby sú jazykovo a platformovo nezávislé, pretože na komunikáciu využívajú XML. To implikuje možnosť použitia nepríbuzných systémov na klientskej a serverovej strane.

2. Druhým dôvodom je použitie protokolu HTTP na prenos správ. Keďže očakávame komunikáciu na úrovni internetu, je vhodné použiť protokol, ktorý nebýva blokovaný na úrovni spojenia.

Samozrejme, využitie webových služieb má aj svoje nedostatky. Hlavnou z nich je prebytočná réžia pri prenose dát. Prenos dát vo forme XML je výrazne náročnejší na šírku prenosového pásma ako dáta v binárnom formáte. Pre aplikácie Gridu sa práve javí byť dôležitejšou práve portabilita, no pre časovo kritické aplikácie bude pravdepodobne nutné hľadať alternatívy.

Samotná architektúra Webových služieb sa zameriava na štyri hlavné oblasti[3].

1. Procesy služieb: táto oblasť sa zaoberá súčinnosťou viacerých služieb. Týka sa napríklad agregovania informácií o prebiehajúcich službách.

2. Opis služby: kľúčovou vlastnosťou webových služieb je schopnosť popísať svoje rozhranie. Toto sa deje prostredníctvom špecializovaného XML jazyka - WSDL (Web Services Description Language). Tento jazyk jednoznačným spôsobom popisuje operácie, ktoré služba okolitému prostrediu poskytuje a ako je možné ju volať.

3. Volanie služby: zahŕňa komunikáciu pomocou správ medzi klientom, ktorý službu volá a serverom, na ktorom je nasadená. Formát v akom sú tieto správy posielané určuje ďalší XML jazyk - SOAP (Simple Object Access Protocol), ktorý vychádza zo skoršieho XML-RPC.

4. Transport: služba sama o sebe nevie prijímať ani odosielať SOAP správy. Používa na to špecializovaný SOAP motor, ktorý správy pre ňu interpretuje. Konkrétne GT4 používa Apache Axis na tieto účely. To však

[3]SOTOMAYOR, B. - CHILDERS, L. 2006. *Globus Toolkit 4*: Programming Java Services. San Francisco : Morgan Kaufmann Publishers, 2006. ISBN-10 0-12-369404-3. s.22

nestačí, pretože Apache Axis nie je serverom sám o sebe, na to potrebuje prostredie, ktoré manažuje prístup viacerých klientov k službe. Bežne používaný aplikačný server, na ktorom je spustený SOAP motor v GT4, je Jakarta Tomcat. Na prenos samotných správ, čo je gro tejto oblasti, je používaný HTTP protokol. Dobrým príkladom, používaným ako štandardný webserver v GT4, je Apache HTTP server.

Sada špecifikácii WSRF hovorí, ako môžeme upraviť webovú službu tak, aby bola stavová. Alebo inak povedané, aby dokázala uchovávať hodnoty premenných, pamätať si svoje predchádzajúce volania. Táto vlastnosť je pre Grid aplikácie kĺúčová, keďže toto požadujeme od väčšiny služieb. Na druhej strane, bezstavovosť nechceme stratiť úplne, keďže v niektorých prípadoch je vhodná.

Riešením je oddelenie samotnej služby od stavu. Stav sa udržuje v zložených premenných zvaných zdroje (resources), z ktorých každý má vlastný kľúč. Ak teda chce klient interagovať s nejakým stavom služby, volá službu konkrétnym kľúčom zdroja. Samozrejme, volanie bez kľúča znamená bezstavovú interakciu.

2.2.1 Vlastnosti zdrojov

Samotné zdroje nie sú najmenšou entitou, v ktorej sa udržujú hodnoty premenných. Tieto sa nazývaju vlastnosti zdroja a každý zdroj ich obsahuje niekoľko, pričom tieto vlastnosti sú definované vo WSDL súbore služby. Vo všeobecnosti môžeme hovoriť o troch typoch vlastností, kroré zdroje obsahujú[4]:

- Dátové hodnoty: sú klasické hodnoty, ktoré by sme očakávali pri tvorbe akejkoľvek aplikácie. Môžu byť nimi výsledky operácii, uchovávané a

[4]SOTOMAYOR, B. - CHILDERS, L. 2006. *Globus Toolkit 4*: Programming Java Services. San Francisco : Morgan Kaufmann Publishers, 2006. ISBN-10 0-12-369404-3. s.35

pomocné premenné výpočtu, alebo vlastnosti služby.

- Metadáta o hodnote: sú pomocné premenné, ktoré pomáhajú spravovať jednotlivé vlastnosti zdroja.

- Metadáta o stave: v mnohom podobné, ako predchádzajúce metadáta, no zaoberajú sa manažmentom zdroja ako celku, nie manažmentom jeho čiastkových vlastností.

2.2.2 WSRF špecifikácia

Formálne je WSRF popísaný v špecifikácii[5], skladajúcej sa zo štyroch častí. Prvá časť, WS-ResourcePropeties, definuje rozhrania, ktoré umožňujú tradičnú priamu prácu s vlastnosťami zdrojov, ako sú čítanie a zápis.

Časť WS-ResourceLifetime pojednáva o životných cykloch zdrojov. Tie totiž nemusia vzniknúť so zapnutím servera, ale môžu vznikať a zanikať počas jeho behu. Systémy, ktoré manažujú takéto živnotné cykly, sú práve predmetom tejto časti.

WS-ServiceGroups poskytuje možnosti ako pracovať so skupinami zdrojov, alebo služieb. Takáto funkcionalita totiž uľahčuje prácu so službami, napríklad prostredníctvom zjednodušeného vyhľadávania. Táto časť obsahuje mechanizmy, ktoré manažujú pridávanie, odoberanie služieb alebo zdrojov zo skupín a vyhľadávanie v ich rámci.

Posledná časť špecifikácie, WS-BaseFaults, rieši základnú réžiu chýb, ktoré vznikajú pri volaní služieb.

[5]CZAJKOWSKI, K. a kolektív. 2004. *The WS-Resource Framework*. Version 1.0. Máj 2004.

2.3 Komponenty Globus Toolkit 4

Ako sme už spomenuli, GT4 je založený na skupine spolupracujúcich a vzájomne sa doplňujúcich služieb. Tieto komponenty sú použiteľné osobitne, ale ich kombináciou je možné vystavať kompletnú GRID aplikáciu. V GT4 rozlišujeme[6] päť základných skupín komponentov v závislosti od ich účelu.

2.3.1 Bezpečnosť

Bezpečnosť je dôležitá v akomkľvek systéme, no v distribuovaných aplikáciách bežiacich vo viacerých lokáciách sú tieto úlohy komplikovanejšie. Väčší počet hráčov chce ovplyvňovať činnosti, ktoré môžu byť v rámci systému vykonávané, vrátane vlastníkov jednotlivých zdrojov, užívateľov, ktorí iniciujú výpočty a virtuálnych organizácií. Grid Security Infrastructure sa skladá zo štyroch komponentov:

- Overovanie a Autorizácia: patria sem nástroje slúžiace k ovládaniu prístupu k službám a zdrojom a nástroje umožnňujúce tvorbu a použitie vlastných autorizačných metód.
- Delegácia: jedným z komponentov je aj služba, ktorá deleguje credentials na kontajner.
- Community Authorization: je skupinová autorizácia pre vytvorené virtuálne organizácie a ich zdroje vo forme služby Community Authorization Service.
- Manažment credentials: obsahuje základnu certifikačnú autoritu GT4 - SimpleCA pre vydávanie X.509 credentials a tiež službu Myproxy, ktorá umožňuje združovanie X.509 Credentials v online repozitári.

[6]FOSTER, I. 2006. *Globus Toolkit Version 4:* Software for Service-Oriented Systems In *Journal of Computer Science and Technology.* vol. 21., no. 4. 2006.

2.3.2 Manažment dát

Spoločným menovateľom Grid aplikácií je správa a integrácia veľkého množstva dát. Dáta musia byť prístupné za podobných podmienok ako samostatná databáza, pričom sú rozmiestnené v rámci virtuálnych organizácií. GT4 poskytuje užitočné nástroje, pomocou ktorých sa dajú konštruovať zaujímavé riešenia s požadovaným výkonom.

- GridFTP poskytuje knižnice a nástroje pre bezpečný, výkonný a spoľahlivý prenos velkých objemov dát. Implementuje rozšírenia ku klasickému protokolu FTP pre využitie v Grid aplikáciách a zachováva pritom schopnosti spolupráce s FTP klientmi a serverom.

- Reliable File Transfer (RFT) je služba založená WSRF, ktorá používa GridFTP na interné prenosy veľkých objemov dát a prináša nové zaujímavé vlastnosti, ako napríklad prerušovanie a opätovné spúšťanie dátových prenosov.

- Replica Location Service (RLS) je škálovateľná služba poskytujúca prístup k informáciám o umiestnení replikovaných súborov a dát.

- Data Replication Service (DRS) používa služby RLS a RFT na vytváranie lokálnych replík v rámci virtuálnych organizácií tak, aby tieto boli k dispozícií používateľom, ktorí s nimi pracujú.

- Globus Data Access and Integration (DAI) je skupina nástrojov vyvinutých v rámci anglického programu eScience za účelom integrácie dát rôznych formátov, ako napríklad XML, alebo relačné databázy.

2.3.3 Vykonávanie

Komponenty tejto kategórie sa zaoberajú s plánovaním, monitorovaním, manažmentom a samotným spúšťaním jednotlivých pracovných jednotiek:

- Grid Resource Allocation and Management (GRAM) je základným komponentom v tejto oblasti. Služba sa používa na zadávanie, monitoring a kontrolu práce na počítačoch. GRAM poskytuje rozhrania pre Unix shell a plánovače Condor, LSF a PBS.

- Plánovače poskytujú GRAM službe sériu úloh na vykonanie pre zhluky, alebo počítače na lokálnej sieti. Patria sem Condor, PBS, LSF, SGE a Torque.

- Community Scheduler Framework (CSF) poskytuje jednotné rozhranie pre dostupné plánovače a unifikuje tak ich ovládanie.

- Choreografia: sem môžeme zaradiť systémy, ktoré podporujú koordináciu vyššieho stupňa implementáciou rôznych modelov paralelných výpočtov. Systémy ako DAGman, MPICH-G2 a ďalšie potom využívajú GRAM na spúšťanie úloh vo vzdialených prostrediach.

2.3.4 Monitorovanie a prieskum

Tieto informačné služby sú kolektívne známe ako Monitoring and Discovery System (MDS). Ich úlohou je zbierať informácie o zdrojoch, diagnostikovať problémy, ktoré môžu vzniknúť pri vykonávaní úloh a identifikovať zdroje, alebo služby s požadovanými vlastnosťami.

- MDS-Index zbiera informácie v XML formáte o zdrojoch a vytvára tak ich register pre virtuálnu organizáciu.

- MDS-Trigger je ďalšia agregačná služba, ktorá na základe charakteru zozbieraných dát spúšťa zvolené akcie.

- WebMDS prezentuje dáta zozbierané agregačnými službami, akými sú aj MDS-Index a MDS-Trigger, v prehľadnej webovej forme pre používateľa.

2.3.5 Spoločná podpora behu

Do tejto kategórie patria komponenty, ktoré poskytujú nástroje a knižnice pre hosťovanie existujúcich služieb a podporu vývoja nových služieb. GT4 poskytuje takéto prostredia pre jazyky Java, C a Python, pričom Java služby sú najfrekventovanejšie, a preto sa nimi budeme aj bližšie zaoberať.

Ako sme už spomenuli, služba na to, aby mohla pracovať, musí byť zahniezdená v tzv. kontajneri, pozostávajúceho zo SOAP motora, aplikačného servera a HTTP servera. GT4 poskytuje jednoduchý samostatný javovský kontajner pre webové služby založený na Apache Axis. Pre bohatšie využitie webových služieb je však možné v GT4 zvoliť pokročilejší aplikačný server, akým je napríklad Apache Jakarta Tomcat.

Kapitola 3

Modelová aplikácia

V nasledujúcom sa pokúsime načrtnúť možnosti, špecifiká a úskalia vývoja grid aplikácií pomocou Globus Toolkitu. Globus Toolkit budeme používať vo verzii 4.0.8, ktorý je označovaný ako overená stabilná verzia. Druhou možnosťou by bolo použiť GT v najnovšej verzii 5.0.0, ktorá však ešte iba teraz vstupuje do svojho životného cyklu, a preto sa predpokladá, že sa v nej objavia chyby. Postupy a riešenia prezentované v tejto kapitole budú však použiteľné aj v nadchádzajúcich verziách.

Pre spomenutý účel budeme vyvíjať modelovú grid aplikáciu, ktorú by bolo možné nasadiť v reálnom prostredí. Naša aplikácia je typovo určená pre menšie lokálne siete, aké je jednoduché vytvoriť aj na akademickej pôde. Hardvér siete teda pozostáva z pracovných staníc v počte do 20. S väčším počtom staníc sa totiž zvyšujú nároky na koordinátora siete v Grid aplikácii. Podmienkou pre každú stanicu je operačný systém Linux, v ktorom je nutné nastaviť prostredie pre danú stanicu. Vhodnou úlohou pre našu Grid aplikáciu je počítanie a overovanie veľkých prvočísel, ktorá dobre reprezentuje typ výpočtov prebiehajúcich na akademických sieťach.

Naša modelová aplikácia využíva tri štandardizované služby, ktoré sú k dispozícii v základnej verzii Globus Toolkitu. Tieto sú základným stavebným

kameňom akejkoľvek Grid aplikácie, preto je vhodné sa nimi zaoberať. Sú nimi Reliable File Transfer (RFT) s pomocou servera GridFTP (odsek 3.1.4), Grid Resource Allocation and Management (WS - GRAM, odsek 3.1.5) a Delegation Service.

3.1 Príprava prostredia

Prvým krokom pred vývojom aplikácie je nastavenie prostredia v každom uzle Gridu, na ktorom bude Globus Toolkit bežať. Najpoužívanejšou platformou pre Globus Toolkit sú linuxové operačné systémy, na ktorých budeme tiež stavať. Za vhodného kandidáta sa bežne považuje distribúcia Red Hat, no my sme zvolili, hlavne kvôli jej rozšírenosti, distribúciu Ubuntu vo verzii 9.04. Ešte pred začatím inštalácie samotného Toolkitu je potrebné nainštalovať nasledujúci sofvér:

- JDK(Java Development Kit): JDK sme použili vo verzii 1.6.1. Okrem samotného vývoja aplikácie je potrebný aj k inštalácii.
- Apache Ant, momentálne vo verzii 1.8.0 je nástroj na zostavovanie spustiteľných súborov používaných v GT, vo svojej podstate nie nepodobný unixovskému príkazu make. Používa takzvaný buildfile, ktorý obsahuje inštrukcie pre Ant o tom, ktoré súbory je potrebné skompilovať, ako ich skompilovať a v akom poradí. Ant vytvorí jediný .GAR súbor pre celú službu, ktorý je potom nasaditeľný do prostredia uzlu.

3.1.1 Inštalácia Globus Toolkit 4.0.8

Pre administráciu a používanie prostredia GT4 je potrebné vytvoriť osobitného užívateľa. Tento užívateľ by nemal mať root práva a mal by mať právo zapisovať iba v adresári určenom pre GT4. Zvolíme teda meno užívateľa ako

globus a adresár ako */usr/local/gt4*. Následne používateľa globus spravíme vlastníkom novovytvoreného inštalačného adresára.

```
root$ adduser globus
root$ mkdir /usr/local/gt4
root$ chown globus.globus /usr/local/gt4
```

Výpis 3.1: Nastevenie globus užívateľa

Toolkit je stiahnuteľný z oficiálnej stránky projektu (http://www.globus-.org/toolkit/downloads). Po rozbalení komprimovaného archívu je možné pristúpiť k samotnej inštalácii. Ako užívateľ globus je potrebné najprv nastaviť premennú prostredia a spustiť inštaláciu:

```
globus$ export GLOBUS_LOCATION=/usr/local/gt4
globus$ ./configure --prefix=$GLOBUS_LOCATION
globus$ make
```

Výpis 3.2: Sputenie GT4 inštalácie

3.1.2 Nastavenie premenných prostredia

Pre zjednodušenie práce pri vývoji aplikácie je vhodné nastaviť premenné užívateľského prostredia a to buď globálne v súbore */etc/profile*, alebo osobitne pre konzolu každého užívateľa v súbore */home/user/.bashrc*. Pre správne fungovanie programov je potrebné nastaviť premenné JAVA_HOME a ANT_-HOME tak, aby ukazovali na inštalačný adresár týchto programov. Podobné je nastavenie premennej GLOBUS_LOCATION. Nakoniec je potrebné zaistiť, aby sa automaticky spúšťal pre užívateľa skript *globus-user-env.sh* a pre vývojára skript *globus-devel-env.sh*, ktoré umiestňujú lokácie knižnice GT4 do premennej CLASSPATH.

```
...
export JAVA_HOME=/usr/bin/jdk-1.6.1/
```

```
export ANT_HOME=/usr/bin/apache-ant-1.7.1/

# GLOBUS
export GLOBUS_LOCATION=/usr/local/gt4/
source $GLOBUS_LOCATION/etc/globus-user-env.sh
source $GLOBUS_LOCATION/etc/globus-devel-env.sh
...
```

Výpis 3.3: Nástroje

3.1.3 Bezpečnosť

Je viacero spôsobov vytvorenia bezpečného prostredia v GT4. My použijeme jednoduchú SimpleCA, ktorá je obsiahnutá v Toolkite. Táto certifikačná autorita (CA) vychádza z OpenSSL. Pred inštaláciou SimpleCA je vhodné synchronizovať systémový čas v uzloch gridu, napríklad pomocou časového serveru NTP.

Jeden z uzlov v sieti zvolíme ako hostiteľa CA. Tento hostiteľ bude vydávať a podpisovať certifikáty pre dva typy používateľov. Prvým sú samotné uzly gridu, ktoré tak majú od certifikačnej autority povolenie byť súčasťou siete, a tiež pre užívateľov Gridu, ktorí sa tak stávajú jeho autorizovanými používateľmi. V tomto hostiteľovi sa prihlásime ako užívateľ globus a spustíme skript *setup-simple-ca*, ktorý nas stručne prevedie nastavením CA. Na konci je vytvorený konfiguračný súbor formátu *globus_simple_ca_(ca_hash)_setup-0.19.tar.gz*, ktorý musí byť prenesený do každého uzla gridu. Nasledujúca séria príkazov najprv ako užívateľ globus, potom ako root, konfiguruje bezpečnosť na uzle vzhľadom k našej CA.

```
[globus@uzol]$ $GLOBUS_LOCATION/sbin/gpt-build globus_simple_ca_(ca_hash)
        _setup-0.19.tar.gz gcc32dbg
[globus@uzol]$ $GLOBUS_LOCATION/sbin/gpt-postinstall
[root@uzol]$ $GLOBUS_LOCATION/setup/globus_simple_ca_(ca_hash)_setup/
        setup-gsi -default
```

Výpis 3.4: Inštalácia certifikčnej autority

Získanie hostiteľského certifikátu

Pre získanie a podpísanie hostiteľského certifikátu v ľubovoľnom uzle gridu je treba podstúpiť nasledujúce kroky:

1. Ako root v každom uzle vytvoríme žiadosť o certifikát;

```
[root@uzol]$ grid-cert-request -host 'menouzla'
```

Výpis 3.5: Vyžiadanie certifikátu pre hostiteľa

2. Vytvorený súbor *hostcert-request.pem* so žiadosťou pošleme hostiteľovi certifikačnej autority a používateľ globus ho následne musí podpísať;

```
[globus@ca]$ grid-ca-sign -in hostcert-request.pem \
-out hostcert.pem
```

Výpis 3.6: Podpis certifikátu pre hostiteľa

3. Podpísaný súbor *hostcert.pem* vrátime naspäť žiadateľovi a prekopírujeme ho do adresára */etc/grid-security/*, ktorý bol vytvorený špeciálne pre účely bezpečnosti gridu

Získanie užívateľského certifikátu

Postup pre získanie užívateľského certifikátu na používanie Grid aplikácie je podobný:

1. Ako root v každom uzle vytvoríme žiadosť o certifikát;

```
[user@uzol]$ grid-cert-request
```

Výpis 3.7: Vyžiadanie certifikátu pre užívateľa

2. Vytvorený súbor *usercert-request.pem* so žiadosťou pošleme hostiteľovi certifikačnej autority a používateľ globus ho následne musí podpísať;

```
[globus@ca]$ grid-ca-sign -in usercert-request.pem \
-out usercert.pem
```

Výpis 3.8: Podpis certifikátu pre užívateľa

3. Podpísaný súbor *hostcert.pem* vrátime naspäť žiadateľovi a prekopírujeme ho do adresára */home/user/.globus/*, ktorý sa nachádza domovskom adresári používateľa;

4. Užívateľský certifikát je možné overiť príkazom:

```
[user@uzol]$ grid-proxy-init -debug -verify
```

Výpis 3.9: Overenie certifikátu

5. Posledným krokom je pridanie záznamu, ktorý mapuje lokálneho užívateľa na certifikovaného užívateľa gridu do súboru */etc/grid-security/grid-mapfile*. V tomto súbore musia byť vymenovaní a namapovaní všetci certifikovaní užívatelia gridu, od ktorých bude uzol prijímať delegované credentials.

3.1.4 GridFTP a Reliable File Transfer - RFT

Tieto grid služby budeme využívať na prenos súborov medzi užívateľmi aplikácie. Tieto služby vyžadujú na svoj beh daemon xinetd a ľubovoľný da-

tabázový server. Pokiaľ uzlová stanica nedisponuje týmto softvérom, je potrebné ho nainštalovať. Ako databázový server sme zvolili PostgreSQL vo verzii 8.3.9. Pre službu GridFTP je potrebné pridať záznam o nej do súboru */etc/services* a vytvoriť pre ňu konfiguračný súbor */etc/xinetd.d/gsiftp* pre daemon xinetd.

RFT bude priamo využívať PSQL server, ktorý je potrebné konfigurovať tak, aby prijímal spojenia zo siete. Všetky potrebné nastavenia je možné spraviť v konfiguračných súboroch PSQL *pg_ hba.conf* a *postgresql.conf*. Spolu s inštaláciou databázového servera sa automaticky vytvorí užívateľ *postgres*. Pomocou tohto užívateľa je potrebné vytvoriť novú databázu pre službu RFT, vytvoriť v nej adekvátne tabuľky pomocou príslušného skriptu a pripustiť užívateľa globus k práci s databázou. Je potrebné aj zmeniť heslo tohto používateľa PSQL tak, aby súhlasilo s heslom používateľa globus operačného systému.

```
[postgres@uzol]$ createdb rftDatabase
[postgres@uzol]$ psql -d rftDatabase -f \
$GLOBUS\_LOCATION/share/globus_wsrf_rft/rft_schema.sql
[postgres@uzol]$ createuser globus
[postgres@uzol]$ \password globus
```

Výpis 3.10: Vytvorenie databázy

3.1.5 Grid Resource Allocation Management - GRAM

Pre konfiguráciu služby WS - GRAM je potrebné iba pridať nasledovné riadky do súboru prislúchajúceho k príkazu sudo. Oba tieto záznamy musia byť uvedené v jednom riadku.

```
...
# Globus GRAM

globus ALL=(user) NOPASSWD:
```

```
/usr/local/gt4/libexec/globus-gridmap-and-execute -g
/etc/grid-security/grid-mapfile
/usr/local/gt4/libexec/globus-job-manager-script.pl *

globus ALL=(user) NOPASSWD:
/usr/local/gt4/libexec/globus-gridmap-and-execute -g
/etc/grid-security/grid-mapfile
/usr/local/gt4/libexec/globus-gram-local-proxy.pl *
...
```

Výpis 3.11: Úprava príkazu sudo

Vo verzii GT 4.0.8 je chyba v užívateľskom nástroji, ktorú je potrebné opraviť pre správne fungovanie delegovania credentials a následne aj služby GRAM. Pre opravu je treba upraviť súbor $GLOBUS_LOCATION/libexec/globus-gram-local-proxy-tool a vymeniť nasledujúcu časť skriptu:

```
...
        # proxyfile should exist
        exec rm "$PROXYFILE"
        exit $?
        ;;
...
```

Výpis 3.12: Chybný zdrojový kód

za:

```
...
        if [ -e "$PROXYFILE" ]; then
                exec rm "$PROXYFILE"
                exit $?
        else
                exit 0
        fi
        ;;
...
```

Výpis 3.13: Opravený zdrojový kód

3.2 Dizajn aplikácie

Dizajn je v Grid aplikáciách kľúčovým krokom. Ešte pred samotným plánovaním aplikácie je potrebné rozhodnúť o možnostiach využitia existujúcich služieb Toolkitu. Tých je pomerne veľké množstvo a v konečnom dôsledku zjednodušujú stavbu aplikácie. Pri implementácii však treba rátať s ich špecifikami, a preto rozhodnutie o ich použití musí byť učinené čo najskôr. Ďalším krokom pri dizajne je modelovanie architektúry. Teda rozdelenie aplikácie do jednotlivých komponentov, rozhodnutie o nasadení týchto komponentov do uzlov siete, definícia ich vzájomnej komunikácie a v konečnom dôsledku spôsob koordinácie uzlov siete. Tretím krokom je výber z bezpečnostných techník, ktoré budeme používať v aplikácii. Zmena týchto techník je totiž uprostred implementácie pomerne komplikovaná.

3.2.1 Využitie služieb

Pri našej aplikácii budeme využívať niekoľko štandardných služieb. Ich účel je nasledovný:

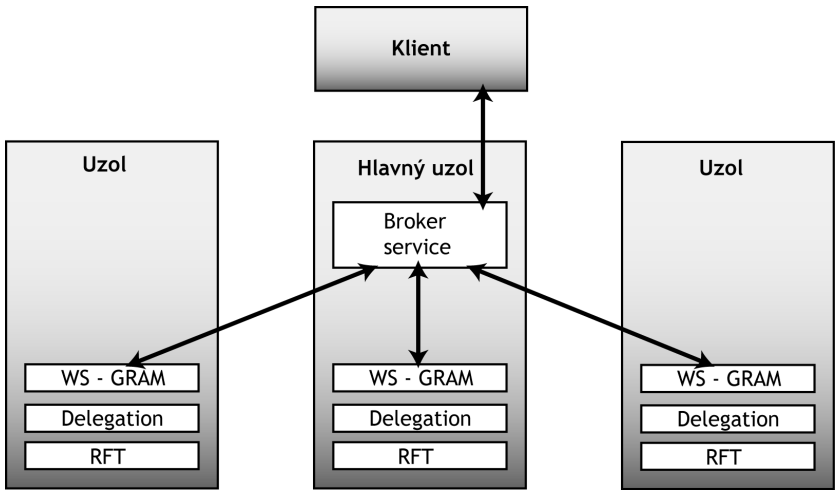
- WS - GRAM: táto služba bude zodpovedná za manažment výpočtov v jedotlivých uzloch Gridu. Využíva službu RFT na prenos dát k uzlu, spustí výpočet, sleduje jeho priebeh a po jeho skončení opäť pomocou služby RFT odovzdá výsledok služby k zadávateľovi. V každom uzle má možnosť využívať niekoľko lokálnych plánovačov na zadávanie úloh. My sme sa pre jednoduchosť rozhodli použiť základnú možnosť, t.j. spúšťanie procesov vetvením Fork.

- Reliable File Transfer: služba RFT bude využívať GridFTP server na transfer zadaní výpočtových úloh do výpočtových úloh a na zasielanie výsledkov naspäť k zadávateľom. Služba nebude použitá priamo, ale prostredníctvom WS - GRAM.

- Delegation Service: výpočet bude spúšťaný v mene užívateľa - zadávateľa výpočtu. Preto potrebujeme službu, ktorá bude delegovať credentials (poverovacie prvky - certifikát a súkromný kľúč) ku kontajnerom uzlov tak, aby tieto prvky mohla použiť služba WS - GRAM pri spustení výpočtu.

3.2.2 Architektúra

Aplikácia bude poskytovať klientovi jediný prístupový bod k zadávaniu úloh pre ľubovoľný počet uzlov. Klient nikdy nebude priamo komunikovať s jednotlivými uzlami. Jeho interakcia bude prebiehať výlučne s vopred zvoleným hlavným uzlom siete, na ktorom bude nasadená naša sprostredkovateľská služba. Táto služba je zodpovedná za komunikáciu s klientom a zároveň všetkými ďalšími výpočtovými uzlami zapojenými do siete. Stáva sa teda koordinátorom celej siete a jej účelom je spúšťať v mene klienta výpočty na uzloch zapojených do siete.



Obr. 3.1: Architektúra systému

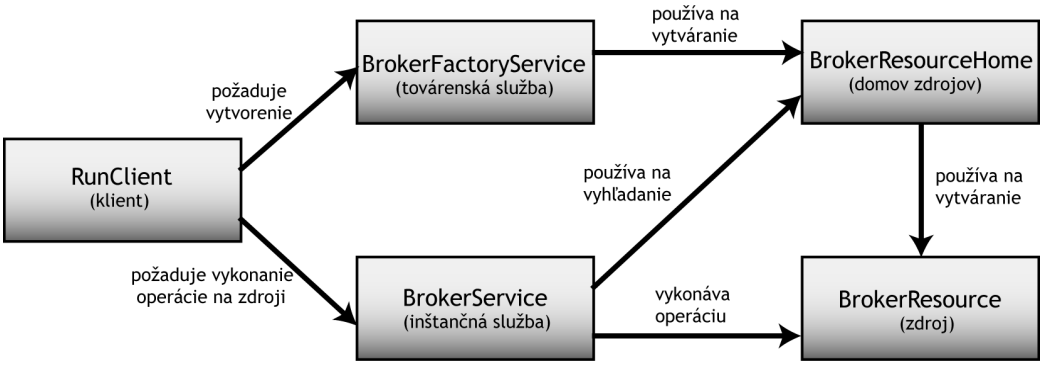
Na každom uzle preto musia byť sprevádzkované všekty tri spomenuté štandardné služby - WS - GRAM, Delegation Service a RFT. Vzhľadom na to, že sieť nebude veľká, bude výpočty vykonávať aj koordinačný uzol so sprostredkovateľskou službou. Réžia nami špecifikovanej siete totiž nebude náročná a zvyšný výkon hlavného uzla bude vhodné využiť na výpočty. V prípade, že by sa inštitúcia nasadzujúca takúto aplikáciu rozhodla pre väčšiu sieť, bolo by vhodné určiť hlavný uzol za uzol venovaný výlučne koordinačnej činnosti.

Úlohou sprostredkovateľskej služby je pomocou troch štandardných služieb delegovať a spúšťať výpočty v uzloch, popri čom si bude držať informácie o jednotlivých výpočtoch a sledovať ich priebeh. Počas priebehu výpočtu, alebo po jeho ukončení, je pripravená odkomunikovať jeho stav klientovi.

3.2.3 Factory-Instance pattern

Jadrom aplikácie je sprostredkovateľská služba Broker Service. V skutočnosti sa však táto služba bude skladať z dvoch spolupracujúcich služieb. Aplikácia totiž musí byť schopná spracovávať viac výpočtových úloh súčasne a ku každej úlohe budeme vytvárať unikátny zdroj. Špecifikácia WSRF odporúča, aby pri práci s viacerými zdrojmi bola ich správa rozdelená medzi továrenskú službu - službu, ktorej úlohou je produkovať nové zdroje (inštancie) a inštančnú službu, ktorá bude so zdrojmi pracovať.

Podľa tohto vzoru bude klient pracovať s obomi službami. Pri zadávaní výpočtu pracuje s továrenskou službou BrokerFactory, ktorá má jedinú operáciu, ktorou vytvára zdroje. Za účelom vytvorenia zdrojov používa domov zdrojov BrokerHome, ktorý sám vytvorí koncový zdroj pre výpočtovú úlohu. Klient bude taktiež kontaktovať inštančnú službu BrokerService, ak bude chcieť vykonať ľubovoľnú operáciu na konkrétnom zdroji. Inštančná služba na základe zdrojového kľúča kontaktuje domov zdrojov, ktorý pre ňu nájde



Obr. 3.2: Factory-instance pattern

požadovaný zdroj a inštančná služba bude následne môcť vykonávať na nájdenom zdroji svoje operácie. V konečnom dôsledku tak môžeme povedať, že domov zdrojov spravuje všetky zdroje tak, aby boli k dispozícii pre obe služby.

3.2.4 Komponenty aplikácie

Celú aplikáciu sme rozdelili na šesť dôležitých implementačných celkov. Teraz si priblížime správanie a účel každého z nich.

Klient `RunClient`

Pomocou tohto klienta bude užívateľ zadávať výpočtové úlohy. Na vstupe je klientskej aplikácii potrebné zadať meno úlohy, cestu k súboru so zadaním úlohy, lokáciu, kam má byť zapísaný výsledok úlohy a identifikačné údaje zadávateľa. Klient RunClient sa pripája na obe služby - BrokerFactoryService aj BrokerService.

Klient `StatusClient`

Úlohou tohto klienta je získať pre užívateľa údaje o konkrétnej úlohe bez ohľadu na to, či ešte prebieha, alebo už bola ukončená. Užívateľ na vstupe identifikuje úlohu, o ktorej chce zistiť podrobnosti iba jej menom. StatusClient kontaktuje za účelom získania informácii iba službu BrokerService.

Inštančná služba `BrokerService`

Táto služba implementuje okrem pomocných metód dve hlavné operácie. Prvou je operácia *run*, ktorá rozhodne o výbere cieľového uzla a na základe dodaných dát od klienta RunClient v ňom spustí vykonávanie úlohy. Druhou je operácia *status*, ktorá vráti klientovi StatusClient informácie o špecifikovanej úlohe. Služba implementuje aj javovské rozhranie *NotifyCallback* podľa špecifikácie WS-Notification, vďaka čomu ju je možné upozorňovať na zmeny stavov jednotlivých úloh počas ich behu.

Továrenská služba `BrokerFactoryService`

BrokerFactoryService je jednoduchá služba s jedinou verejnou operáciou *createResource*. Na žiadosť klienta RunClient vytvára nový zdroj BrokerResource a vracia koncovú referenciu k tomuto zdroju.

Domov zdrojov `BrokerResourceHome`

Je rozšírením štandardného riešenia domova zdrojov v GlobusToolkite - triedy ResourceHomeImpl. Táto trieda je rozšírená o metódu *create*, ktorá okrem vytvorenia zdroja BrokerResource vytvorí aj korešpondujúci kľúč ku zdroju a vráti ho.

Zdroje `BrokerResource`

Zdroj BrokerResource je informačným zdrojom ku každej úlohe. Vzniká zároveň so zadaním úlohy od klienta, no svoju informačnú úlohu by mal plniť aj po skončení úlohy v časovom horizonte dní. Preto požadujeme, aby zdroje vydržali aj reštarty Gridu. Riešením je perzistencia všetkých zdrojov v hlavnom uzle siete. Zoznam sledovaných vlastností úlohy a teda aj zdroja pozostáva z nasledovných položiek:

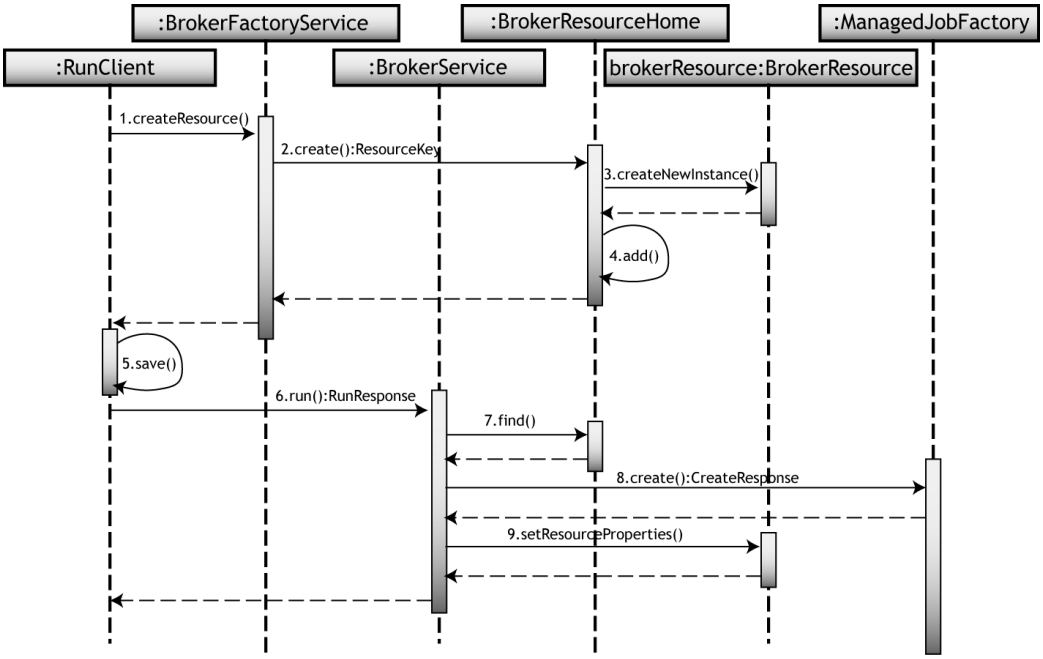
gramJobHandleRP - unikátny identifikátor úlohy
conditionRP - posledný kontrolovaný stav úlohy
inputUriRP - adresa súboru zadania úlohy
outputUriRP - adresa súboru výsledku úlohy
stationRP - uzol, v ktorom bola úloha spustená
userNameRP - užívateľovo meno
userEmailRP - užívateľov kontaktný e-mail
dateSubmissionRP - dátum zadania úlohy
stdErrorRP - adresa súboru pre chybový výstup
stdOutputRP - adresa súboru pre štandardný výstup
jobNameRP - meno úlohy podľa užívateľa

3.2.5 Sekvenčný diagram - `Run`

Prvou operáciou, ktorú si rozoberieme je operácia Run.

1. Klient RunClient najprv kontaktuje službu BrokerFactoryService a pomocou operácie *createResource()* žiada o vytvorenie nového zdroja. Služba vracia koncovú referenciu typu *EndpointReference* k novovytvorenému zdroju.

2. Služba potom pomocou triedy ResourceContext nájde domov BrokerResourceHome. Po vyhľadaní vyvolá jej metódu *create()*, ktorou si vy-



Obr. 3.3: Sekvenčný diagram operácie `Run`

žiada vytvorenie nového zdroja. Táto operácia vráti po vytvorení služby kľúč ResourceKey k tomuto zdroju.

3. Domov vytvorí nový zdroj ako novú inštanciu triedy BrokerResource pomocou konštruktora *createNewInstance()*.

4. Nakoniec si domov pridá nový zdroj aj s novým kľúčom do svojej internej mapy zdrojov pomocou metódy *add()*.

5. Klient RunClient si uloží koncovú referenciu na kombináciu inštančnej služby BrokerService a konkrétneho zdroja brokerResource pre ďalšiu prácu.

6. Následne klient spustí s touto referenciou operáciu *run()* inštančnej služby BrokerResource. Návratová hodnota je typu RunResponse, ktorá

obsahuje informácie o úspešnosti operácie. Pritom klient dodá všetky potrebné informácie na spustenie výpočtovej úlohy.

7. Podľa koncovej referencie za pomoci triedy ResourceContext vyhľadá služba BrokerService konkrétny zdroj metódou *find()* domova BrokerResourceHome.

8. Služba má v tomto bode k dispozícii ako potrebné dáta k úlohe, tak aj korešpondujúci nový zdroj. Preto kontaktuje službu ManagedJobFactory (WS - GRAM) v niektorom z uzlov a vytvorí u nej výpočtovú úlohu operáciou *create()*. Táto vracia odpoveď typu CreateResponse.

9. Po overení spustenia úlohy nastaví služba BrokerService hodnoty korešpondujúceho zdroja brokerResource pomocou operácií typu *setResourceProperty()* a predá odpoveď RunResponse klientovi.

3.2.6 Sekvenčný diagram - `Status`

Druhou, jednoduchšou operáciou je Status.

1. Klient si podľa mena zvolí úlohu, ktorej stav chce zistiť. StatusClient potom zavolá operáciu *status()* s koncovou referenciou na konkrétny zdroj prislúchajúci k úlohe. Návratovou hodnotou tejto operácie je StatusResponse, v ktorej sú uložené aktuálne informácie o prebiehajúcej úlohe.

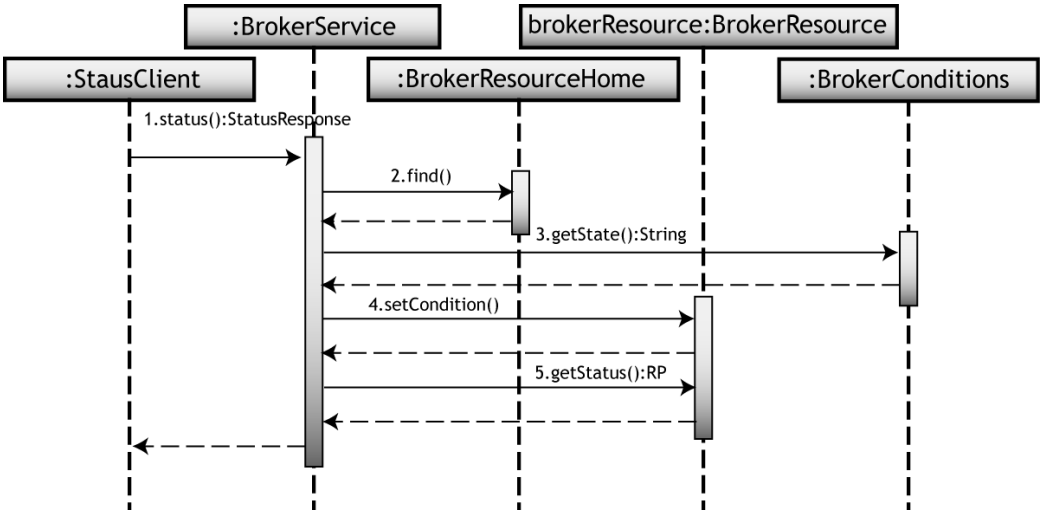
2. Služba BrokerService si vyžiada od domova BrokerResourceHome správny zdroj operáciou *find()* pomocou triedy ResourceContext.

3. Potom sa obráti na pomocnú triedu BrokerConditions a vyvolá metódu *getState()*, ktorá odpovie súčasným stavom vykonávanej úlohy. Pomocná trieda BrokerConditions má za úlohu sledovať prebiehajúce

zmeny stavov všetkých úloh, o ktorých je služba BrokerService notifikovaná.

4. Nový zistený stav uloží do brokerResource pomocou metódy *setCondition()*.

5. Následne si metódou typu *getResourceProperty()* vypýta hodnotu uloženú v brokerResource a vloží ju do odpovede StatusResponse určenej pre klienta StatusClient.



Obr. 3.4: Sekvenčný diagram operácie `status`

3.3 Implementácia

V predchádzajúcej podkapitole sme zvážili dôležité aspekty stavby aplikácie. Po naplánovaní teda môžeme prejsť k implementačnej fáze projektu. Aj keď by služba mohla byť implementovaná aj v jazykoch C++, alebo Python, zvolíme jazyk Java, ktorý je použitý aj vo väčšine celého Toolkitu. Napísanie

a nasadenie akejkoľvek služby musí prebiehať v GT vždy v nasledujúcich krokoch, podľa ktorých budeme postupovať.

Definícia rozhrania: Na definíciu rozhrania je určený špeciálny XML jazyk - Web Service Description Language (WSDL). Účelom tohto súboru je zverejniť vlastnosti a schopnosti služby. Budú v ňom definované operácie, ktoré služba poskytuje klientom, formát správ pre komunikáciu a vlastnosti zdrojov prislúchajúcich k službe. Na základe tohto súboru budeme generovať javovské stuby, ktoré bude klient používať pre využívanie možností služby.

Implementácia služby: Zdrojový kód bude rozdelený na niekoľko osobitných tried podľa návrhu. Implementácia musí spĺňať náležitosti definované vo WSDL súbore.

Nastavenie parametrov nasadenia: V tomto kroku spresníme parametre, podľa ktorých bude služba spúšťaná v kontajneri. Docielime to pomocou dvoch súborov. Jedným je Web Service Deployment Descriptor (WSDD), ktorý definuje okrem iného aj adresu, kde bude služba dostupná. Druhým je JNDI súbor, ktorý bude obsahovať nastavenie prístupu k zdrojom.

Kompilácia a nasadenie: Celý zdrojový kód skompilujeme pomocou nainštalovaného nástroja ANT. Ten vytvorí balík pre celú službu vo formáte .gar. Na nasadenie použijeme ďalší nástroj s názvom globus-deploy-gar, ktorý sa postará o rozbalenie balíka do správnych lokácií GT4 inštalácie.

3.3.1 Rozhranie `BrokerPortType`

Potrebujeme napísať rozhrania pre obe služby - BrokerService, aj BrokerFactoryService. Obe tieto rozhrania sú uložené v osobitných súboroch Bro-

ker.wsdl a Factory.wsdl. Rozhranie pre továrenskú službu je však triviálne, a preto sa budeme venovať len komplikovanejšiemu - rozhraniu pre BrokerService.

Rozhranie je rozdelené na tri časti. Prvou je všeobecná deklarácia rozhrania, tzv. PortType. Rozhodli sme sa využiť štandardizované postupy práce s vlastnosťami zdrojov, ktoré sú v GT4 implementované. Bolo by možné definovať aj vlastný spôsob prístupu k vlastnostiam, no štandardizáciou umožníme prístup aj klientovi kompatibilnému so štandardom WSRF, nie len nášmu špecifickému. Preto naše rozhranie služby BrokerService - BrokerPortType bude rozširovať existujúce PortTypy, ktoré umožňujú prístup k vlastnostiam zdrojov štandardizovanými get/set metódami. Tiež budeme štandardným spôsobom pristupovať aj k deštrukcii zdrojov - budeme mať možnosť zdroj zničiť buď na vyžiadanie, alebo podľa vopred rozhodnutého načasovania.

V PortType taktiež uvádzame operácie, ktoré rozhranie, a teda aj služba poskytuje - operácie run a status. Ku každej operácii priraďujeme správu, ktorá ide k operácii, resp. ktorá je operáciou vracaná.

```
<portType name="BrokerPortType"
        wsdlpp:extends="wsrpw:GetResourceProperty
                    wsrpw:GetMultipleResourceProperties
                    wsrpw:SetResourceProperties
                    wsrpw:QueryResourceProperties
                    wsrlw:ImmediateResourceTermination
                    wsrlw:ScheduledResourceTermination
        wsrp:ResourceProperties="tns:BrokerResourceProperties">

        <operation name="run">
                <input message="tns:RunInputMessage"/>
                <output message="tns:RunOutputMessage"/>
        </operation>
        <operation name="status">
                <input message="tns:StatusInputMessage"/>
                <output message="tns:StatusOutputMessage"/>
        </operation>
</portType>
```

Výpis 3.14: BrokerPortType

V druhej časti deklarujeme jednotlivé správy prislúchajúce k operáciám, ktoré definujeme v tretej časti. Operácia run má dve správy - RunInputMessage, ktorej prislúcha element správy run a RunOutputMessage, ktorej prilúcha element správy runResponse. Podobne, operácii status prislúchajú vstupná a výstupná správa StatusInputMessage a StatusOutputMessage s analogicky definovanými elementmi. Službe posielame pri príkaze na spustenie úlohy inštancie komplexných typov run a status. Typ run pozostáva z dát potrebných na spustenie, zatiaľ čo status je prázdny, keďže pre vyžiadanie informácií od zdroja nepotrebujeme posielať žiadne informácie službe.

PortType obsahuje okrem všetkých ostatných správ aj zoznam definícií vlastností zdroja služby BrokerService. Práve k tomuto zoznamu budeme môcť pristupovať pomocou štandardizovaných metód, ktoré nám PortTypy, z ktorých sme rozšírili náš BrokerPortType, umožňujú. Je potrebné však dodať, že v našej aplikácii budeme priamo využívať iba metódy vyplývajúce zo základných PortTypov GetResourceProperty a ScheduledResourceTermination. Avšak vďaka štandardizácii budú mocť klientske nástroje GT4 využívať aj metódy poskytnuté ostatnými PortTypmi, z ktorých sme náš BrokerPortType rozširovali. Preto bude možné napríklad ľahko ničiť ktorýkoľvek zo zdrojov uložených v hlavnom uzle, alebo robiť dopyty na zdroje pomocou jazyka XPath.

3.3.2 Domov `BrokerResourceHome`

Väčšinu funkcionality, ktorú by sme očakávali od správcu zdrojov, už implementuje trieda ResourceHomeImpl. Táto trieda obsahuje kolekciu pre zdroje, metódy na vyhľadávanie, ničenie a pridávanie zdrojov. Budeme z nej teda vychádzať, vytvoríme jej novú metódu *create()*, ktorá vytvorí nový zdroj a

vráti zdrojový kľúč. Na vytvorenie kľúča použijeme pomocnú triedu UUIDGenFactory, ktorá nám zaručí vytvorenie unikátneho kľúča. Práve s týmto vytvoreným kľúčom ako identifikačným číslom môžeme inicializovať zdroj. Samotný zdrojový kľúč, ktorý BrokerResourceHome zaradí do svojej mapy, však musí byť objekt typu SimpleResourceKey, ktorý sa vytvára z hodnoty key ľuboboľného kľúča spolu s typom kľúča keyTypeName. Zvyšok implementácie BrokerResourceHome je priamočiary.

```
Object key = (Object)UUIDGenFactory.getUUIDGen().nextUUID();
brokerResource.initialize(key);
ResourceKey rkey = new SimpleResourceKey(keyTypeName, key);
```

Výpis 3.15: Vytvorenie kľúča

3.3.3 Služba `BrokerFactoryService`

Implementácia továrne je veľmi jednoduchá. Ako sme už spomínali, táto služba je zodpovedná za vytváranie nových zdrojov pomocou jedinej operácie *createResource()*. Preto, aby túto funkciu mohla plniť, bude využívať pomocnú triedu ResourceContext, ktorá má za úlohu poskytnúť abstrakciu od problému zistenia identifikátora zdroja a jeho následného vyhľadania. Po získaní kontextu z neho získa domov pomocou metódy *getResourceHome()* domov, ktorý spravuje zdroje. Náš domov je však odvodený, preto musíme návratový objekt metódy pretypovať. Potom získame z domova pri samotnom vytvorení nového zdroja kľúč daného zdroja.

```
context = ResourceContext.getResourceContext();
home = (BrokerResourceHome) context.getResourceHome();
key = home.create();
```

Výpis 3.16: Vytvorenie nového zdroja

Tento kľúč spolu s URI inštančnej služby tvorí dokopy koncovú referenciu, vďaka ktorej bude môcť klient k tomuto zdroju opäť pristupovať. Preto musíme túto koncovú referenciu odovzdať klientovi ako návratovú hodnotu tejto operácie. URI poskladáme z reťazcov hostiteľa služby, lokality služby a adresy inštančnej služby. Nakoniec vytvoríme koncovú referenciu metódou *createEndpointReference()* z URI a získaného kľúča nového zdroja.

```
URL baseURL = ServiceHost.getBaseURL();
String instanceService = (String) MessageContext.getCurrentContext().
        getService().getOption("instance");
String instanceURI = baseURL.toString() + "project/first/"+ instanceService;
epr = AddressingUtils.createEndpointReference(instanceURI, key);
```

Výpis 3.17: Konštrukcia koncovej referencie

3.3.4 Zdroj `BrokerResource`

V časti dizajn sme určili, že zdroj BrokerResource musí byť schopný zotavenia po páde servera, preto jeho implementáciu staviame na triede PersistentReflectionResource. Úlohou zdroja je spravovať svoje vlastnosti, ktoré sme taktiež načrtli v návrhu. Pre všetky tieto účely sme implementovali nasledujúcich šesť javovských rozhraní GT:

Resource - má za úlohu iba identifikovať triedu ako triedu pre zdroj, nesľubuje žiadne metódy.

ResourceIdentifier - zaväzuje nás implementovať metódu *getID()*, ktorou by sa mal každý zdroj byť schopný identifikovať. Naším identifikátorom, ktorý vracia táto metóda, je unikátny kľúč, ktorý prideľuje BrokerResourceHome.

ResourceProperties - označuje našu triedu, že musí obsahovať sadu vlastností ResourcePropertySet. Do tejto triedy uložíme všetky naše vlast-

nosti a zároveň sme pre ne vytvorli get/set metódy. Tieto get/set metódy sú určené pre použitie v službe BrokerService, pretože klienti k nim budú pristupovať prostredníctvom štandardizovaných operácií vyplývajúcich z rozšírenia rozhrania BrokerPortType.

ResourceLifetime - chceme zničiť zdroj sedem dní po skončení úlohy zadanej pre GRAM. Preto využijeme rozhranie, ktoré implikuje pridanie vlastnosti terminationTime do zdroja, ktorá určuje čas jeho zničenia. Zároveň nás núti implementovať metódy *getTerminationTime()* a emphsetTerminationTime(), ktoré pracujú s týmto časom.

PersistenceCallback - sľubuje implementáciu dvoch metód, ktoré sú jadrom perzistencie. Metóda *store()* má za úlohu uložiť zdroj na disk. Uloženie sa týka všetkých vlastností zdroja, ktoré sú v ResourcePropertySet. Prvýkrát bude volaná hneď pri vytvorení, potom vždy pri zmene informácií v zdroji. Druhou je *load()*, ktorá je vždy automaticky volaná domovom BrokerResourceHome, ak sa daný zdroj nenachádza v pamäti a je ho potrebné načítať z disku.

RemoveCallback - implementujeme metódu *remove()*, ktorá bude mať na starosti vymazanie zdroja z disku. Použijeme naň pomocnú triedu FilePersistenceHelper.

3.3.5 Služba `BrokerService`

Táto trieda je jadrom celej Grid aplikácie a využíva všetky doteraz spomenuté triedy. Jej úlohou je poskytovať klientom operácie *run()* a *status()*, ktorými užívateľ manažuje svoju výpočtovú úlohu. Z pohľadu mechaniky táto služba ovláda ďalšie tri štandardné služby Globus Toolkitu - službu GRAM, službu Delegation Service a nepriamo cez GRAM aj RFT.

Vzorovou výpočtovou úlohou je simulovaný výpočet v poradí n-tého prvočísla. Pre tento účel používame jednoduchý program Primes.java[1], ktorý načíta zadanie čísla n zo súboru a ktorého výstupom je inkriminované n-té prvočíslo. Predpokladom je, že tento program sa v skompilovanej podobe nachádza nainštalovaný na všetkých uzloch výpočtového Gridu.

Operácia `run`

Operácia run, ktorú sme definovali rozhraní Broker.wsdl služby BrokerService je implementovaná ako verejná metóda triedy BrokerService. Keď užívateľ pomocou klienta RunClient zavolá spustenie novej výpočtovej úlohy, spustí sa táto metóda. Táto, aby mohla použiť službu GRAM, jej musí najprv skonštruovať zadanie úlohy.

GRAM prijíma zadanie úlohy v XML jazyku zvanom RSL. Naše zadanie sa však musí meniť v závislosti od dát, ktoré dostala metóda na vstupe. Preto neprichádza do úvahy nahranie vopred pripraveného RSL súboru. Naše zadanie teda skonštruujeme pomocou parametrizovanej metódy *createRSL()*, ktorá zo vstupu od klienta vytvorí RSL reťazec so zadaním úlohy pre vybraný uzol. Naše zadanie je nastavené tak, aby spúšťalo predinštalovaný program Primes, no je treba podotknúť, že jednoduchou zmenou v RSL by bolo možné zadanie rozšíriť na akýkoľvek spustiteľný súbor, ktorý by mal vykonávať výpočet.

Gros metódy je však v nastavení potrebných služieb, spojení a bezpečnosti tak, aby sa výpočet mohol spustiť. Odohráva sa v niekoľkých krokoch:

1. Ako prvé je potrebné pripojiť sa na rozhranie (Port) služby ManagedJobFactory, čo je továrenská služba GRAM. V celom procese pomáha trieda ManagedFactoryClientHelper, ktorá disponuje užitočnými metódami *getServiceURL()*, *getFactoryEndpoint()* a *getPort()*. Pri napájaní

[1] zdrojový kód je možné nájsť v prílohe B

sa na rozhranie je potrebné aj určiť typ továrne, t.j. lokálny plánovač pre úlohy. V našom prípade bol použitý základný Fork.

2. V predchádzajúcom kroku sme sa pripájali na službu GRAM v cieľovom uzle pre úlohu. Potrebujeme však získať aj koncovú referenciu na Delegation službu bežiacu v tom istom uzle. Tá bude mať na starosti delegovanie credentials.

3. Potom je možné prejsť na nastavenie bezpečnosti. Naša služba BrokerService bude v role klienta k serverovej strane, na ktorej je GRAM. Pre túto povahu služby musíme najprv skonštruovať klientsky popis bezpečnosti typu ClientSecurityDescriptor. Na úrovni prenosu informácii použijeme šifrovanie, čo je štandardná prenosová bezpečnosť GT. Pre vedecké výpočty by boli ďalšie možnosti so šifrovaním na úrovní zbytočne zaťažujúce. Nakoiec nastavíme pre popis bezpečnosti aj používané credentials klienta.

```
HostAuthorization iA = new HostAuthorization();
ClientSecurityDescriptor secDesc = new ClientSecurityDescriptor();
secDesc.setGSITransport(Constants.ENCRYPTION);
secDesc.setAuthz(iA);
secDesc.setGSSCredential(cred);
```

Výpis 3.18: Nastavenie bezpečnostného opisu

Následne je možné credentials delegovať do uzla a vytvoriť koncovú referenciu credentialEndpoint na tieto delegované prvky. Credentials bude totiž využívať aj služba RFT, ktorú GRAM použije na prenos súborov. Preto jej parametre nastavíme tak, aby používala credentials z koncovej referencie credentialEndpoint.

4. Ďalším krokom je samotné spustenie úlohy v uzle definovanom v kroku (1). Služba GRAM poskytuje na tento účel operáciu *createManaged-*

Job(), ktorej vstupom je objekt typu CreateManagedJobInputType. Tento objekt obsahuje všetky informácie, ktoré GRAM potrebuje k vykonaniu úlohy, ak už vopred boli zaručené práva na vykonanie prostredníctvom credentials. Vstupu jobInput priradíme nový unikátny identifikačný údaj z továrne UUIDGenFactory, nastavíme mu maximálny čas behu v úlohy a priradíme mu zadanie získané načítaním RSL reťazca.

```
CreateManagedJobInputType jobInput = new CreateManagedJobInputType();
jobInput.setJobID(new AttributedURI("uuid:" +
        UUIDGenFactory.getUUIDGen()));
jobInput.setInitialTerminationTime(cal);
jobInput.setJob(jobDescription);
CreateManagedJobOutputType createResponse =
        factoryPort.createManagedJob(jobInput);
```

Výpis 3.19: Zadanie úlohy

5. Po spustení si vypýtame identifikačný reťazec k práve spustenej úlohe typu gramJobHandle. Týmto reťazcom budeme vedieť jednoznačne identifikovať úlohu prislúchajúcu k niektorému zdroju brokerResource. Preto tento reťazec aj zaraďujeme medzi vlastnosti zdroja.

Operácia `status`

Táto operácia je relatívne jednoduchá a jej najdôležitejšou úlohou je skontrolovať stav prijatej notifikácie o úlohe. Ak táto notifikácia prebehla, zapíše ju do vlastnosti nášho zdroja pomocou metódy *BrokerResource.setCondition()* a uloží premennú na disk pomocou *BrokerResource.store()*.

Notifikácie

Služba GRAM je konfigurovaná podľa špecifikácie WS-Notification. To znamená, že je implementovaná ako producent notifikácií. Zdroje služby GRAM ponúkajú poskytovanie notifikácií o zmenách niektorých vlastností. Nás bude

zaujímať zmena stavu výpočtovej úlohy. Tá počas svojho životného cyklu prechádza niektorými z nasledujúcich stavov: Unsubmitted, StageIn, Pending, Active, Suspended, StageOut, CleanUp, Done a Failed. Kontaktovanie úlohy počas jej behu je problematické a nešikovné, preto je jednoduchšie sa pri štarte úlohy v metóde *run()* registrovať u služby GRAM ako prijímateľ notifikácií a iba zachytávať správy o zmene stavu.

Na to, aby sa BrokerService mohla stať prijímateľom notifkácií, ku ktorým je zaregistrovaná, musí implementovať javovské rozhranie NotifyCallback a implementovať metódu *deliver()*, ktorá je volaná vždy v momente príchodu notifikačnej správy.

```
...
else if (message instanceof StateChangeNotificationMessageWrapperType) {
        notif = ((StateChangeNotificationMessageWrapperType) message).
                getStateChangeNotificationMessage();
}
state = notif.getState().getValue();
```

Výpis 3.20: Zachytenie notifikačnej správy

Správne zachytenie správy vyžaduje identifikáciu prichádzajúceho typu správy. Tá totiž môže prísť okrem StateChangeNotificationMessageWrapperType aj ako typ Element. Po správnej identifikácii typu správy je z nej možné vybrať notifikáciu metódou *getStateChangeNotificationMessage()*. Potom už nie je problém získať nový stav úlohy vo forme reťazca.

3.3.6 Klienti `RunClient` a `StatusClient`

Obe klientske aplikácie majú mnoho spoločných prvkov, ktoré sa zároveň musia vyskytovať aj v akomkoľvek klientovi určenom pre Grid aplikáciu postavenú v GT4. U oboch sme implementovali rovnakú metódu *printResourceProperties()*, ktorá využíva pre získanie vlastností zdrojov metódy štandardných PortTypov, na ktorých sme náš BrokerPortType postavili.

Jediným prístupovým bodom k službám BrokerService a BrokerFactoryService sú ich korešpondujúce rozhrania, teda PortTypy. Pri kompilácii celej aplikácie sa vygenerujú z .wsdl súborov javovské stuby, ktoré klienti používajú. Preto aj akýkoľvek vzdialený klient, ktorý bude chcieť využiť službu a operácie, ktoré poskytuje, bude musieť mať prístup k stubom rozhrania.

Ako bolo načrtnuté v dizajne, klienti musia používať koncové referencie na prístup k úlohám. Tieto koncové referencie sa skladajú z adresy inštančnej služby a identifikátora, čiže kľúča zdroja, ktorý výpočtovej úlohe prináleží. Preto, aby mohli klienti opakovane pristupovať k úlohám a kontrolovať ich stavy, je potrebné si ich zapísať. Koncové referencie teda zapisujeme do súborov na lokalitu, ku ktorej má užívateľ Gridu s právami spúšťať úlohy prístup. Na zápis a čítanie takýchto objektov na disk využijeme implementované triedy Globusu ObjectSerializer a ObjectDeserializer.

```
FactoryBrokerServiceAddressingLocator factoryLocator =
        new FactoryBrokerServiceAddressingLocator();
factoryEPR = new EndpointReferenceType();
factoryEPR.setAddress(new Address(factoryURI));
brokerFactory = factoryLocator.getFactoryPortTypePort(factoryEPR);
...
brokerFactory.createResource(new CreateResource());
```

Výpis 3.19: Spojenie sa so službou

Najdôležitejším krokom je však kontaktovanie služby, ktorej operácie chceme využívať. To sa deje prostredníctvom lokátorov, ktoré sú vygenerované v rámci javovských stubov z .wsdl súboru. Ku každej službe je vygenerovaná špeciálna lokalizačná trieda, v prípade BrokerFactoryService je to FactoryBrokerServiceAddressingLocator. K službe pristúpime cez koncovú referenciu, ktorú buď načítame zo súboru pomocou triedy ObjectDeserializer, alebo ako v prípade BrokerFactoryService rovno cez adresu. BrokerFactoryService totiž nemá žiadne vlastné zdroje a koncová referencia teda pozostáva iba zo samostatnej adresy. Cez továrenský lokátor potom pomocou metódy *getFac-*

toryPortTypePort() získame rozhranie na kontaktovanú službu. Cez toto rozhranie je potom možné jednoducho volať všetky definované operácie z .wsdl rozhrania. Postup pri práci s inštančnou službou je analogický.

3.3.7 Nasadenie

Ako sme už spomenuli, nastavenie nasadenia sa odohráva dvoch v XML súboroch, ktorých mená sú povinné. Prvým je deploy-jndi-config.xml a obsahuje nastavenia parametrov práce so zdrojmi ako sú formát kľúča zdroja, meno kľúča zdroja, alebo identifikácia továrenskej služby.

Druhým je deploy-server.wsdd so samotným nastavením služieb. Okrem štandardných nastavení web služieb, ako identifikácia .wsdl súboru k službe, identifikácia súboru s bezpečnostným nastavením, alebo rozsah povolených metód sem patrí aj parameter poskytovateľov (name=“providers“).

V rozhraní k službe BrokerService sme deklarovali, že naša služba implementuje štandardizované WSRF PortTypy. Týmito rozhraniami boli GetResourceProperty, GetMultipleResourceProperties, SetResourceProperties a QueryResourceProperties pre prístup k dátam, a ImmediateResourceTermination, ScheduledResourceTermination pre deštrukciu zdrojov. Avšak táto deklarácia vo .wsdl súbore znamená, že určité štandardné metódy budú implementované. Pre funkcionalitu musíme vo .wsdd súbore služby dodať aj operačných poskytovateľov týchto metód. V našom prípade to budú GetRPProvider, GetMRPProvider, SetRPProvider a QueryRPProvider pre dopytovacie PortTypy a DestroyProvider a SetTerminationTimeProvider pre deštrukčné PortTypy. Títo poskytovatelia sú v skutočnosti skratky pre existujúce javovské triedy, ktoré implementujú sľubované metódy.

3.3.8 Bezpečnosť

V rámci bezpečnosti služby dokážeme identifikovať dva vzťahy. V prvom vzťahu vystupuje služba BrokerService ako klient služby GRAM (ManagedJobFactoryService), ktorá v mene BrokerService spúšťa výpočtové úlohy. Nastaveniu zabezpečenia sme sa už stručne venovali v časti o samotnej službe. Druhým, zjavným vzťahom je klientska strana klientov RunClient a StatusClient, pričom úlohu servera hrá služba BrokerService. U tohto vzťahu potrebujeme nastaviť obidve strany tak, aby brali do úvahy bezpečnostné charakteristiky prostredia.

Naším úmyslom bude delegovať credentials z klienta na službu tak, aby služba mohla pracovať v mene užívateľa z klientskej strany. Služba bude môcť potom tieto prvky delegovať ďalej na službu GRAM v uzloch. Touto dvojstupňovou delegáciou budeme vedieť pre každý uzol určiť, ktorý užívateľ Gridu má k zadávaniu úloh prístup. Na to, aby sme povolili delegovanie credentials na strane klientov, potrebujeme zvoliť metódu zabezpečenia prenosu. Na výber máme z možností GSISecureMessage, GSISecureConversation, ktoré poskytujú zabezpečenie na úrovni správ a GSITransport, ktoré poskytuje možnosť zabezpečenia na transportnej úrovni. Uprednostníme metódu GSISecureConversation pred GSITransport, keďže volaní na tejto úrovni neočakávame až taký veľký počet, aby sa kvôli zvýšenému prenosu dát plynúceho z metódy zabezpečenia zahltila sieť. Delegácia credentials je tiež možná len pri využití tejto metódy. Povolíme plné delegovanie credentials a zabezpečíme ho digitálnym podpisom tagom <integrity>.

```
<GSISecureConversation>
        <integrity/>
        <delegation value="full"/>
</GSISecureConversation>
```

Výpis 3.21: Bezpečnostný opis klienta

Na strane služby BrokerService potrebujeme nastaviť bezpečnostnú konfiguráciu pre operácie run a status. Nastavenie sa odohráva, rovnako ako v prípade klientov, v špeciálnom XML súbore s bezpečnostným opisom. V našom prípade budeme používať rovnaké zabezpečenie u oboch operácií.

Služba pri volaní operácie run bude prijímať správy zabezpečené metódou GSISecureConversation vo forme s digitálnym podpisom, alebo šifrovaním. Naši klienti budú posielať správy iba s digitálnym podpisom, no ak by sme sa v budúcnosti rozhodli implementovať ďalšieho klienta s iným typom zabezpečenia, nebudeme musieť kvôli jeho sfunkčneniu meniť aj službu.

Druhým prvkom nastavenia je značka run-as, ktorá určí, čie credentials budú použité. Naším cieľom je beh s právami užívateľa Gridu, preto nastavíme tento element na caller-identity. O správnu delegáciu credentials sa potom postará Globus automaticky.

```
<method name="run">
        <auth-method>
                <GSISecureConversation>
                        <protection-level>
                                <integrity/>
                                <privacy/>
                        </protection-level>
                </GSISecureConversation>
        </auth-method>
        <run-as>
                <caller-identity/>
        </run-as>
</method>
```

Výpis 3.22: Bepečnostný opis služby

3.4 Zhodnotenie programátorských praktík

Globus Toolkit poskytuje veľmi bohaté možnosti tvorby Grid aplikácií každého druhu. Vďaka konceptom webových služieb a zdrojov je možné postaviť

v podstate ľubovoľnú architektúru aplikácie. Je zrozumiteľne a jednoznačne vymedzený spôsob komunikácie medzi službami a klientmi pomocou rozhraní definovaných vo formáte XML. Táto platformová nezávislosť jazyka rozhraní umožňuje stavbu projektov z komponentov vyvinutých v rôznych jazykoch. Ďalším plusom tohto prístupu je modularita komponentov v takto vyvíjanej aplikácii. Je pomerne jednoduché vyvinúť ďalšieho klienta len na základe rozhrania k službe, či nechať novú službu používať ľubovoľnú inú.

K spomínanej modularite prispievajú aj existujúce štandardné služby, ktoré sú k dispozícii pre vývoj komplexných aplikácií. Ich repertoár pokrýva základné oblasti bezpečnosti, správy dát, vykonávania a informačných služieb. Z týchto komponentov je možné vyskladať aplikácie s oveľa nižšími časovými nárokmi, ako by si vyžadoval vývoj celej aplikácie od základov. Samozrejme, ak ľubovoľný komponent svojou funkcionalitou nevyhovuje, má programátor možnosť použiť vlastné komponenty. Na druhej strane však treba s použitím štandardných komponentov počítať hneď od začiatku projektu a aplikáciu je potrebné im prispôsobiť, čo pre niektoré aplikácie môže byť nevyhovujúce.

Okrem štandardných služieb a komponentov bol vyvinutý veľký počet ďalších komponentov v rámci rôznych Grid projektov, ktoré sú voľne dostupné a použiteľné v akejkoľvek aplikácii. Celý projekt je vedený ako open-source, čo poskytuje obrovské možnosti pre vývojára. V prípade potreby je totiž možné prevziať časť kódu už existujúceho komponentu, alebo si prispôsobiť komponent tak, aby vyhovoval navrhnutej aplikácii. Ďalej táto povaha projektu Globus Toolkit umožňuje učenie sa vývojárov priamo na zdrojovom kóde Toolkitu. Pre programátora je jednoduché pozrieť sa na zdrojový kód používaného komponentu, zistiť jeho implementačné detaily a podľa toho sa zariadiť pri implementácii aplikácie.

Zároveň je však pre Globus Toolkit otvorenosť zdrojových kódov aj nutnosťou. Úroveň dokumentácie z pohľadu vývojára nedosahuje potrebnú úro-

veň. Samozrejme, existujú výpisy aplikačných rozhraní všetkých implementovaných tried, podľa ktorých sa dá zbežne orientovať. No ako sme sa mali možnosť presvedčiť, programovanie služieb v GT4 má svoje špecifické postupy. Aplikácia sa často skladá zo sérií príkazov, pričom každá séria má za úlohu vykonať čiastkovú úlohu. Príkladom môže byť štandardný spôsob získania vlastností zdrojov, pripojenie sa na rozhranie služby, či postup pri volaní služby GRAM. Problémom je, že tieto postupy je potrebné dodržať a zároveň sa skladajú z volaní metód relatívne veľkého počtu rôznych tried. Z popisov rozhraní teda nie je možné získať poznatky o správnych postupoch volaní metód a nie je možné získať potrebný ucelený obraz o postupe. Počet tutoriálov venujúcich sa tejto problematike je veľmi obmedzený a preto vývojárovi nezostáva nič iné ako inšpirovať sa postupmi v zdrojových kódoch existujúcich komponentov.

Ďalším úskalím je nutnosť rozdelenia aplikácie do relatívne veľkého počtu súborov. Pritom relatívne veľká časť z nich sú konfiguračné súbory v rôznych XML formátoch. Aj menší projekt, ako ten náš, obsahuje deväť konfiguračných súborov s nastavením rôznych parametrov, funkcionalít a mapovaní. Preto je aj problematické hľadanie chýb a ladenie programu, keďže chyba nemusí byť len v samotnom zdrojovom kóde, ale aj v nastaveniach.

Inštaláciu Globus Toolkitu môžeme tiež považovať za ďalšiu problematickú pasáž, ktorá by si zaslúžila pozornosť Globus Alliace. Toolkitu totiž po základnej inštalácii chýba ešte relatívne veľa k plnej funkčnosti. Ak pominieme nastavenie certifkačnej autority a potrebných certifikátov, ktoré sú samozrejmou nutnosťou, každá zo štandardných služieb vyžaduje rozsiahlu dodatočnú konfiguráciu. Často využívajú externé nástroje, ktoré je potrebné doinštalovať, pretože nie sú súčasťou Globus balíka. Konfigurácie samotných služieb vyžadujú zásahy do dôležitých systémových súborov, či nástrojov. Preto je plné nastavenie systému relatívne náročnou úlohou, čo zvyšuje časové nároky na nasadenie a zároveň aj nároky na administrátora konkrétneho

uzla.

V oblasti bezpečnosti a ochrany informácií je Globus Toolkit na veľmi vysokej úrovni. Poskytuje širokú paletu možností nastavenia bezpečnosti. Kombinácia X.509 certifikátov a 3 módov šifrovania predstavuje silnú kombináciu pre overovanie identity užívateľov a zabezpečenie dátových prenosov. Bohaté sú tiež možnosti autorizácie prístupu overených užívateľov k rôznym službám. Kombináciou týchto metód je možné zabezpečiť ľubovoľne flexibilný Grid systém.

Možnosti rozšírenia modelovej aplikácie sú jasné. Globus Toolkit poskytuje všetky potrebné prvky na to, aby sa náš projekt dal nasadiť do reálneho produkčného prostredia. Okrem potrebnej bohatšej réžie na strane klientov a širšej ponuky operácií na strane služby BrokerService, je možné integrovať ďalšie komponenty Toolkitu. Vhodné by bolo použitie indexovej služby v spolupráci s WebMDS, ktoré by nám mohli poskytnúť prehľadné informácie o vyťaženosti jednotlivých uzlov. Aplikácia je nastavená na jednu konkrétnu lokálnu sieť. Použitím indexovej služby by bolo možné dosiahnuť automatickú konfiguráciu Grid aplikácie v prostredí ľubovoľnej lokálnej siete. Ďalším užitočným krokom by mohlo byť zavedenie lokálnych plánovačov v uzloch, akým je napríklad PBS.

Záver

Odborný časopis Masachusettského technologického inštitútu zaradil Grid computing a konkrétne Globus Toolkit medzi „desať nastupujúcich technológií, ktoré zmenia svet“[2]. Na objektívne posúdenie tejto predpovede ešte budeme musieť počkať, no už teraz je zrejmé, že využitie Grid technológií rapídne rastie. V oblasti vedeckých aplikácii vytláčajú kolaboratívne Gridy centralizované superpočítače pri zlomkových nákladoch na ich obstaranie a prevádzkovanie. Veľké Gridy svojou výpočtovou silou, či objemom spracovaných dát už dávnejšie predčili najvýkonnejšie superpočítače. Okrem uplatnenia v oblasti výskumu však Grid nachádza uplatnenie aj v ďalších sférach, ktorým sme sa stručne venovali v úvode práce.

Väčšina Gridov súčasnosti pritom používa Globus Toolkit a jeho komponenty pre stavbu aplikácií. Špecifikáciám, na ktorých je Toolkit postavený a ich vzťahom sme sa venovali v druhej časti. Čitateľ sa mal taktiež možnosť oboznámiť so základnými komponentmi, ktoré Toolkit tvoria.

V tretej časti sme sa venovali implementácii modelovej služby v Globus Toolkite. Naším cieľom bolo demonštrovať postupy potrebné na vytvorenie aplikácie a posúdiť efektívnosť Toolkitu pri tvorbe Grid aplikácie v menších lokálnych sieťach. Toolkit svojou ponukou štandardných komponentov ponúka výborné možnosti pre stavbu menších služieb, no má aj svoje záporné

[2] *10 Emerging Technologies That Will Change the World* In *An MIT enterprise technology review*. 2003.

stránky. Časovo náročné nastavenie jednotlivých uzlov pre beh, či slabšia dokumentácia povinných programových mechanizmov existujúcich tried a služieb zvyšujú nároky na zručnosti a skúsenosti vývojárov, čo môže predstavovať ťažkosti v menších Grid projektoch. Úžitok Toolkitu je však veľký a tieto problémy je možné časom prekonať, preto niet pochýb o tom, že Globus Toolkit je vhodný aj pre aplikáciu na malých lokálnych sieťach.

Globus Toolkit v prvom kvartáli roka 2010 podstúpil prechod na verziu 5.0.0. Skladá sa predovšetkým z inkrementálnych vylepšení jestvujúcich komponentov. No niektoré sa dočkali výraznejších zmien. GRAM5 bol kompletne prepísaný, aby umožnil lepšiu škálovateľnosť a zlepšil spoľahlivosť, ktorá vo verzii 4 zaostávala. Externá funkcionalita a rozhrania však ostávajú zachované, čo by malo zjednodušiť migráciu starších systémov na verziu 5. Otázka, ako sa bude Toolkit vyvíjať a prisôsobovať novým požiadavkám v budúcnosti, je stále otvorená, no je pravdepodobné, že aj naďalej ostane štandarom rýchlo sa rozvíjajúceho odvetvia Grid computingu.

Literatúra

[1] SOTOMAYOR, B. - CHILDERS, L. 2006. *Globus Toolkit 4*: Programming Java Services. San Francisco : Morgan Kaufmann Publishers, 2006. ISBN-10 0-12-369404-3

[2] FOSTER, I. - KESSELMAN, C. 2004. *The Grid: Blueprint for a New Computing Infrastructure.* 2. vydanie. San Francisco : Morgan Kaufmann Publishers, 2004. ISBN 1-55860-9-334

[3] FOSTER, I. - KESSELMAN, C. 1998. *Computational Grids.* 1998.

[4] GLOBAL GRID FORUM. 2005. *The Open Grid Services Architecture, Version 1.0.* 2005.
`http://www.gridforum.org/documents/GFD.30.pdf`

[5] CZAJKOWSKI, K. a kolektív. 2004. *The WS-Resource Framework.* Version 1.0. Máj 2004.
`http://www.globus.org/wsrf/specs/ws-wsrf.pdf`

[6] FOSTER, I. - KESSELMAN, C. - NICK, J.M. - TUECKE, S. 2002. *The Physiology of the Grid:* An Open Grid Services Architecture for Distributed Systems Integration. 2002.
`http://www.globus.org/alliance/publications/ papers/ogsa.pdf`

[7] JACOB, B. a kolektív. 2003. *Enabling Applications for Grid Computing with Globus*. 1. vydanie. IBM RedBooks, 2003. ISBN 0738453331 `http://www.redbooks.ibm.com/redbooks/pdfs/sg246936.pdf`

[8] JACOB, B. - BROWN, M. - FUKUI, K. - TRIVEDI, N. 2005 *Introduction to Grid Computing*, 1. vydanie. IBM RedBooks, 2005. ISBN 0738494003 `http://www.redbooks.ibm.com/redbooks/pdfs/sg246778.pdf`

[9] FOSTER, I. - KESSELMAN, C. - TUECKE, S. 2001. *The Anatomy of the Grid:* Enabling Scalable Virtual Organizations. 2001. `http://www.globus.org/alliance/publications/papers/anatomy.pdf`

[10] FOSTER, I. 2006. *Globus Toolkit Version 4:* Software for Service-Oriented Systems In *Journal of Computer Science and Technology*. vol. 21., no. 4. 2006.

[11] FOSTER, I. 2002. *What is the Grid? A Three Point Checklist* In *GridToday*. Júl, 2002.

[12] WANKAR, R. *Grid Computing with Globus: An Overview and Research Challenges* In *International Journal of Computer Science and Applications*. vol. 5, no. 3.

[13] MATYSKA, L. 2008. *Empowering Grids – the EGEE gLite middleware* 2008.

[14] ANDERSON, D.P. 2004. *BOINC:* A System for Public-Resource Computing and Storage. In /em 5th IEEE/ACM International Workshop on Grid Computing, strany 365 - 372, 2004.

[15] ANDERSON, D.P. - REED, K. 2008. *Celebrating Diversity in Volunteer Computing*. 2008. `http://boinc.berkeley.edu/boinc_papers/hicss_08/ hicss_08.pdf`

[16] LUTHER, A. - BUYYA, R. - RANJAN, R. - VENUGOPAL, S. *Alchemi:* A .NET-based Enterprise Grid Computing System. 2006.
`http://www.buyya.com/papers/alchemi_icomp05.pdf`

[17] NADIMINTI, K. - BUYYA, R. 2006 *Global Grid Computing Where Are We Today?* In *Enterprise Open Source Journal*, November / December 2006.

[18] ENTERPRICE GRID ALLIANCE, 2005. *Enterprise Grid Security Requirements.* 2005.
`http://ogf.org/documents/ega-grid_security`
`_requirements-v1Approved.pdf`

[19] CHAPPELL, D. 2008. *A Short Introduction to Cloud Platforms:* An Enterprise-oriented View. 2008.
`http://www.davidchappell.com/CloudPlatforms-Chappell.pdf`

[20] YEO, C.S. a kolektív. 2007. *Utility Computing on Global Grids.* 2007.
`http://www.buyya.com/papers/HandbookCN_Utility_Grids.pdf`

[21] DE ROURE, D. - JENNINGS, N.R. - SHADBOLT, N.R. 2005. *The Semantic Grid: Past, Present, and Future.* In *Proceedings of the IEEE.* vol. 93, no. 3. 2005.
`http://www.semanticgrid.org/documents/`
`semgrid2004/semgrid2004.pdf`

[22] LEWIS, M.J. a kolektív, 2003. *Support for Extensibility and Site Autonomy in the Legion Grid System Object Model.* 2003.
`http://www.cs.virginia.edu/papers/JPDC03.pdf`

[23] EGEE Technical Infrastructure
`http://technical.eu-egee.org/index.php?id=134`

[24] *10 Emerging Technologies That Will Change the World* In *An MIT enterprise technology review*. 2003.
http://www.technologyreview.com/Infotech/13060/

[25] UNICORE
http://www.unicore.eu/

[26] GLite
http://glite.web.cern.ch/glite/

[27] Condor
http://www.cs.wisc.edu/condor/

[28] Globus Toolkit
http://www.globus.org/toolkit/

[29] GT 4.0.x Quickstart Guide
http://www.globus.org/toolkit/docs/4.0/admin/
docbook/quickstart.html

[30] Installing GT 4.0 (System Administrator's Guide)
http://www.globus.org/toolkit/docs/4.0/admin/
docbook/admin.pdf

[31] GT 4.0 WS_GRAM
http://www.globus.org/toolkit/docs/4.0/execution/
wsgram/index.pdf

[32] Submitting a job in Java using WS GRAM
http://www.globus.org/toolkit/docs/4.0/execution/
wsgram/WS_GRAM_Java_Scenarios.html

[33] https://cgsrv1.arrc.csiro.au/subversion/apacgrid/

Prílohy

Príloha A - Slovník pojmov

Grid - mriežka, jeden systém zložený z viacerých pracovných staníc
scientific Grid - výskumný kolaboratívny grid systém
public resource computing - výpočty na verejných zdrojoch
utility computing - úžitkové výpočty, výpočty na vyžiadanie
enterprise computing - podnikové výpočty
middleware - sprostredkovateľský softvér
cluster - zhluk, zoskupenie serverov, alebo staníc
daemon - program bežiaci na pozadí
open source - softvér s otvoreným zdrojovým kódom
credentials - poverovacie prvky (bezpečnostný certifikát a súkromný kľúč)
PortType - rozhranie webovej služby definované pomocou WSDL
factory-instance pattern - vzor továreň-inštancia
tag - značka

Príloha B - Trieda Primes

Výpočstovým programom, ktorý spúšťa GRAM v uzloch je jednoduchá trieda na výpočet n-tého provčísla. Samozrejme existujú oveľa efektívnejšie algoritmy, tento však bol zvolený kvôli jeho pamäťovej nenáročnosti.

```
public class Primes {
        //default stop at 100 million
        public static final long DEFAULT_STOP = 100000000;
        //first prime
        public static final int FPRIME = 2;

        public static void main(String[] args) {

        long[] prime = new long[2];
        long stop = DEFAULT_STOP;
        String taskFile = args[0];

        try {
                File file = new File(taskFile);
                FileReader inputFile = new FileReader(file);
                BufferedReader in = new BufferedReader(inputFile);

                String s = in.readLine();
                stop = Long.parseLong(s);
                in.close();
        } catch (Exception e) {
                e.printStackTrace();
        }

        prime[0] = 1;
        prime[1] = FPRIME;

        long n = FPRIME+1;
        int j = 1;
        boolean isPrime = true;

        do {
                if (isPrime) {
                        prime[0] = ++j;
                        prime[1] = n;
                        isPrime = false;
```

```
                }
                n += 2;

                for (int i = 3; i <= n; i=i+2) {
                        long a = n / i;
                        long b = n % i;
                        if (b == 0) {
                                break;
                        }
                        if (a <= i) {
                                isPrime = true;
                                break;
                        }
                }
        } while (j < stop);

        System.out.println("The "+ prime[0] +"-th prime is " + prime[1]);

  }
}
```

Primes.java

Príloha C - Štruktúra projektu

Obsah adresára GRID:

```
./build.mappings
./build.xml
./globus-build-service.py
./globus-build-service.sh
./namespace2package.mappings
./Primes.java
./org/globus/project/clients/BrokerService_instance/RunClient.java
./org/globus/project/clients/BrokerService_instance/StatusClient.java
./org/globus/project/clients/BrokerService_instance/Client_security.xml
./org/globus/project/services/first/deploy-jndi-config.xml
./org/globus/project/services/first/deploy-server.wsdd
./org/globus/project/services/first/etc/security-config.xml
./org/globus/project/services/first/impl/BrokerConditions.java
./org/globus/project/services/first/impl/BrokerConstants.java
./org/globus/project/services/first/impl/BrokerFactoryService.java
./org/globus/project/services/first/impl/BrokerResource.java
./org/globus/project/services/first/impl/BrokerResourceHome.java
./org/globus/project/services/first/impl/BrokerService.java
./schema/project/broker/Broker.wsdl
./schema/project/broker/Factory.wsdl
```

build.mappings - navádzací súbor pre kompiláciu s lokáciou služieb a wsdl súborov

build.xml - súbor s inštrukciami pre nástroj Ant, obsahuje postup pre vytvorenie balíka .gar

globus-build-service.py - kompilačný skript v jazyku python

globus-build-service.sh - kompilačný skript

namespace2package.mappings - mapovací súbor pre preklad XML namespaces

Primes.java - simulačná trieda hľadania n-tého prvočísla, viď príloha B

RunClient.java - trieda RunClient, implementácia klienta pre zadávanie úloh

```
class RunClient {
        void printResourceProperties(BrokerPortType broker);
        void main(String[] args);
}
```

StatusClient.java - trieda StatusClient, implementácia klienta pre zisťovanie stavu úloh

```
class StatusClient {
        void printResourceProperties(BrokerPortType broker);
        void main(String[] args);
}
```

Client_security.xml - bezpečnostný opis pre klientov RunClient a StatusClient

deploy-jndi-config.xml - nastavenie spôsobu použitia zdrojov službami

deploy-server.wsdd - nastavenie nasadenia služieb BrokerService a BrokerFactoryService

security-config.xml - bezpečnostný opis pre nastavenie služieb servera

BrokerConditions.java - pomocná trieda pre správu notifikácií

```
class BrokerConditions {
        String getCondition(int i);
        String getHandle(int i);
        void addCondition(String handle, String condition);
        int find(String handle);
        String getState(String handle);
}
```

BrokerConstants.java - rozhranie obsahujúce konštanty QNames

BrokerFactoryService.java - trieda BrokerFactoryService, implementácia továrenskej služby

```
class BrokerFactoryService {
        CreateResourceResponse createResource(CreateResource request);
}
```

BrokerResource.java - trieda BrokeResource, implementácia zdroja

```
class BrokerResource
        extends PersistentReflectionResource
        implements Resource, ResourceIdentifier, ResourceLifetime,
                   ResourceProperties, PersistenceCallback, RemoveCallback {
        Class getResourceBeanClass();
        Object getID();
        ResourcePropertySet getResourcePropertySet();
        Object initialize(Object key);
        void setTerminationTime(Calendar time, boolean storeFlag);
        Calendar getTerminationTime();
        Calendar getCurrentTime();
        String getGramJobHandle();
        void setGramJobHandle(String gramJobHandle, boolean storeFlag);
        String getCondition();
        void setCondition(String condition, boolean storeFlag);
        String getInputUri();
        void setInputUri(String inputUri, boolean storeFlag);
        String getOutputUri();
        void setOutputUri(String outputUri, boolean storeFlag);
        String getStdOutput();
        void setStdOutput(String stdoutfile);
        String getStdError();
        void setStdError(String stderrfile);
        String getJobName();
        void setJobName(String jobName);
        void setJobName(String jobName, boolean storeFlag);
        String getStation();
        void setStation(String station, boolean storeFlag);
        String getUserName();
        void setUserName(String userName, boolean storeFlag);
        String getUserEmail();
        void setUserEmail(String userEmail, boolean storeFlag);
        Calendar getDateSubmission();
        void setDateSubmission(Calendar c, boolean storeFlag);
```

```
        void load(ResourceKey key);
        void store();
        void remove();
}
```

BrokerResourceHome.java -trieda BrokerResourceHome, implementácia domova zdrojov

```
class BrokerResourceHome extends ResourceHomeImpl {
        ResourceKey create();
        Set retrieveAllKeys();
}
```

BrokerService.java -trieda BrokerService, implementácia inštančnej služby

```
class BrokerService implements NotifyCallback {
        BrokerResource getResource();
        BrokerResourceHome getHome();
        BrokerResource selectResource(String jobHandle);
        void updateConditions();
        String selectStation(String[] inputStations);
        GSSCredential getUserCredential();
        String createRSL(String name, String stdOut, String stdErr, String inURI,
                String outURI, String station);
        void logSecurityInfo(String methodName);
        void deliver(List topicPath, EndpointReferenceType producer,
                Object message);
        RunResponse run(Run runInstance);
        StatusResponse status(Status statusInstance);
}
```

Broker.wsdl - WSDL rozhranie pre službu BrokerService

Factory.wsdl - WSDL rozhranie pre službu BrokerFactoryService