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

Diplomová práce

Správa zdrojů výpočetního gridu

Plzeň, 2005 František Maroušek

Prohlášení

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

V Plzni dne 24.8.2005, František Maroušek

Poděkování

Na tomto místě bych rád poděkoval svým rodičům za poskytnuté zázemí a podporu, dále Ing. Ladislavu Pešičkovi za vedení celé práce, pomoc, cenné rady a připomínky, a všem ostatním, kteří mi pomáhali při studiu.

Abstract

Target of this diploma thesis is to implement example interaction between grid system and mobile clients. The first part of thesis contains theoretical background of grids. The next chapter is focused on Globus Toolkit (version 3 and 4) and on creation of services in GT4 (chapter 4). Chapter 5 describes different approaches to interaction of grid system and mobile clients. Final chapter contains the description of implementation, which is realized throught two simple services - WeatherService and IndexService.

Obsah

1 Úvod **4**

2 Úvod do technologie gridů **5**

2.1 Překlad slova Grid 5
2.2 Vznik gridu 5
2.3 Definice Gridu 6
2.4 Vývoj gridových technologií 7
2.5 Grid vs. cluster 8
2.6 Organizace zabývající se gridy 9
2.6.1 Vývoj gridových standardů 9
2.6.2 Vývoj grid toolkitů, frameworků a middleware 10
2.6.3 Implementace grid technologií ve společnostech 10
2.6.4 Implementace grid technologií v komerční sféře 12
2.7 Struktura gridu 12
2.8 OGSA 15
2.9 OGSI 17
2.10 WSRF 17
2.11 Další informace o gridech 19

3 Globus Toolkit **20**

3.1 Historie Globus Toolkitu 20
3.2 GT3 21
3.2.1 Jádro GT3 22
3.2.2 Základní služby GT3 - Security 23
3.2.3 Základní služby GT3 - Data Management 24

3.3 GT4 27

4 Služby v GT4 **30**
4.1 Co je to služba 30
4.2 Struktura služby v GT4 30
4.2.1 Rozdíl v implementaci služeb mezi GT3 a GT4 32
4.3 GRAM a MDS 33

5 Mobilita **34**
5.1 Mobilní přístup ke gridům 34
5.2 Možnosti interakce klient-grid 35
5.2.1 Rozhoduje služba 36
5.2.2 Rozhoduje klient 37

6 Implementace **40**
6.1 Implementační model 40
6.1.1 Proces činnosti modelu 42
6.1.2 Zpracování chyb 43
6.2 Služba PocasiService 43
6.2.1 Soubor MarService.wsdl 43
6.2.2 Soubory *.java 46
6.2.3 deploy-server.wsdd 48
6.2.4 deploy-jndi-config.xml 49
6.3 Služba IndexService 49
6.4 Klienti 50
6.4.1 IndexClient.java 51
6.4.2 MarClient.java 51
6.5 Testování a výsledky 52

7 Závěr **56**

A Soubory služby a překlad **63**

B Uživatelská příručka **66**

Seznam obrázků

2.1 Vývoj gridových technologií 8
2.2 Condor a Globus 11
2.3 Struktura gridu 13
2.4 Architektura gridů 16
2.5 WS-Resource 18

3.1 Struktura GT3 21
3.2 Vztah mezi OGSA, OGSI a GT3 22
3.3 Reliable File Transfer 26
3.4 RLS 26
3.5 Struktura GT4 28
3.6 Vztah mezi GT4, OGSA a WSRF 29

4.1 Implementace služby v GT4 31
4.2 Service Data Element 33

5.1 Nalezení optimální služby jinou službou 36

6.1 Implementační model 41

B.1 GUI pro vyváření zdrojů pro PocasiService 68
B.2 GUI pro dynamickou aktualizaci zdrojů 68

Kapitola 1

Úvod

Předkládaná diplomová práce se zabývá návrhem a realizací jednoduché interakce gridu s mobilním klientem. Problematika gridů je velmi rozsáhlá, a proto je práce zaměřena pouze na dílčí oblasti z gridových technologií.

Gridy představují další evoluční krok ve vývoji počítačových sítí. V posledních letech se s jejich pomocí řeší čím dál tím více problémů, na které běžné výpočetní kapacity nestačí, ale nejsou používány pouze k výpočetním úkonům. Najdou uplatnění téměř ve všech oblastech, kde je důležité sdílení zdrojů (výpočetních, úložných, atd.). Jejich obrovskou výhodou je, že využívají již existující infrastrukturu, a proto se stávají velmi silným prostředkem pro řešení mnoha problémů, které byly do této doby za rozumnou cenu neřešitelné.

Druhá kapitola se zabývá obecným přehledem z oblasti gridů. Je zde definice, vývoj gridů, nejdůležitější organizace a projekty zabývající se gridy. Na konci této kapitoly je popsána jedna ze struktur gridu a je zde uveden přehled standardů, které jsou pro gridy důležité.

Třetí kapitola se zabývá popisem Globus Toolkitu, který se stal defacto standardem na poli gridových technologií. Popsány jsou verze 3 a 4.

Kapitola 4 obsahuje teorii, která se váže k vytváření služeb v Globus Toolkitu 4.

V kapitole 5 je podrobně rozebráno několik případů, jak by mohla vypadat interakce gridu s mobilním klientem.

Realizace jednoho takového případu je popsána v kapitole 6.

Kapitola 2

Úvod do technologie gridů

Tato kapitola obsahuje obecné informace o gridu jako takovém, jeho struktuře a organizacích zabývající se výzkumem a praktickou relalizací gridů.

2.1 Překlad slova Grid

Dle slovníku by se slovo grid dalo přeložit jako *mřížka*. Toto slovo vzniklo vyjmutím ze sousloví *electrical power grid*. Překlad těchto slov je v českém jazyce *elektrizační soustava*, *elektrická síť* nebo *elektrická rozvodná síť/soustava*. Pokud bychom se při překladu slova grid drželi analogie s elektrickou sítí (případně plynovou, vodovodní sítí, atd.), tak by byly nejvýstižnějšími návrhy pro překlad například:

- výpočetní rozvodná soustava
- výpočtovodní řad
- výpočtovod

Z výše uvedeného vyplývá, že slovo grid nemá cenu překládat do českého jazyka, a proto bude dále používán původní anglický název.

2.2 Vznik gridu

Na počátku minulého století se začala masově využívat elektřina. Nejprve se zaváděla v jednotlivých domech napojených na generátory. Poté se domy začaly propojovat do

větších sítí, aby byla výroba a spotřeba elektřiny efektivnější a aby nedocházelo k jejímu plýtvání. Někde jí vyráběl generátor více, než bylo třeba, a někde se jí naopak nedostávalo. Po určitém čase vznikla elektrická distribuční síť, která byla rozprostřena po celém státě a následně i kontinentu. V této síti je několik zdrojů elektřiny a dochází k jejímu efektivnímu využívání dle požadavků v jednotlivých částích sítě.

Z tohoto příkladu lze odvodit rozvoj počítačových sítí. Od jednotlivých počítačů přes malé sítě k rozsáhlým mezikontinentálním sítím. Bohužel uvedené uspořádání není ideální z hlediska efektivního využívání zdrojů v těchto sítích. V některých částech je například vysoce nevyužita výpočetní kapacita sítě a někde se jí nedostává. Z příkladu elektrické sítě je vidět, že gridy, jakožto sítě, které dokáží využít plný potenciál současných počítačových sítí, jsou pouze dalším evolučním krokem ve vývoji těchto sítí.

2.3 Definice Gridu

První gridy začaly vznikat v roce 1998, kdy se pánové Carl Kesselman a Ian Foster pokusili o počáteční definici gridu [1]:

> *Výpočetní grid je hardwarová a softwarová infrastruktura, která poskytuje spolehlivý, standardizovaný, všudypřítomný a levný přístup ke špičkovým výpočetním službám.*

V době vytvoření této definice gridu se neuvažovalo o přístupu na vyžádání (on-demand) ke službám, datům a výpočetní kapacitě. Proto v roce 2000 vznikla druhá definice gridu [3]:

> *Gridové prostředí je spojeno s koordinovaným sdílením zdrojů a řešením problémů v dynamických, multi-institucionálních virtuálních organizacích.*

Klíčová myšlenka gridu je schopnost vytvořit dohody o sdílení zdrojů mezi několika zůčastněnými stranami (poskytovateli a spotřebiteli) a poté tyto zdroje využít k vyřešení určitého problému. Sdílením zdrojů se nemyslí pouze výměna souborů, ale spíše přímý přístup k počítačům, softwaru, datům a jiným zdrojům dle potřeby řešeného problému. Toto sdílení musí být samozřejmě plně kontrolováno, což znamená přesně určit, kdo a jaké zdroje sdílí, kdo a za jakých podmínek může využívat sdílené zdroje.

Instituce, které jsou definovány výše uvedenými pravidly o sdílení zdrojů tvoří *virtuální organizace (VO)*. Tyto VO se mohou navzájem lišit v účelu, rozsahu, velikosti, době existence, struktuře, atd... Rozdíl mezi VO a reálnými organizacemi je v tom, že v jedné VO může být začleněno několik reálných organizací.

Pojmy jako sdílení zdrojů a VO nejsou nové myšlenky. Například v roce 1969 Len Kleinrock předpověděl [4], že v budoucnosti vzniknou v IT služby, které se budou rozvíjet podobně jako služby v oblasti telefonie a elektřiny a budou sloužit domovům a kancelářím v celé zemi.

Grid lze tedy definovat jako systém, který koordinuje distribuované zdroje za použití standardních, otevřených a k všeobecnému použití vytvořených protokolů a rozhraní, které slouží k doručení netriviálních QoS[2].

Grid musí tedy splňovat následující 3 kritéria:

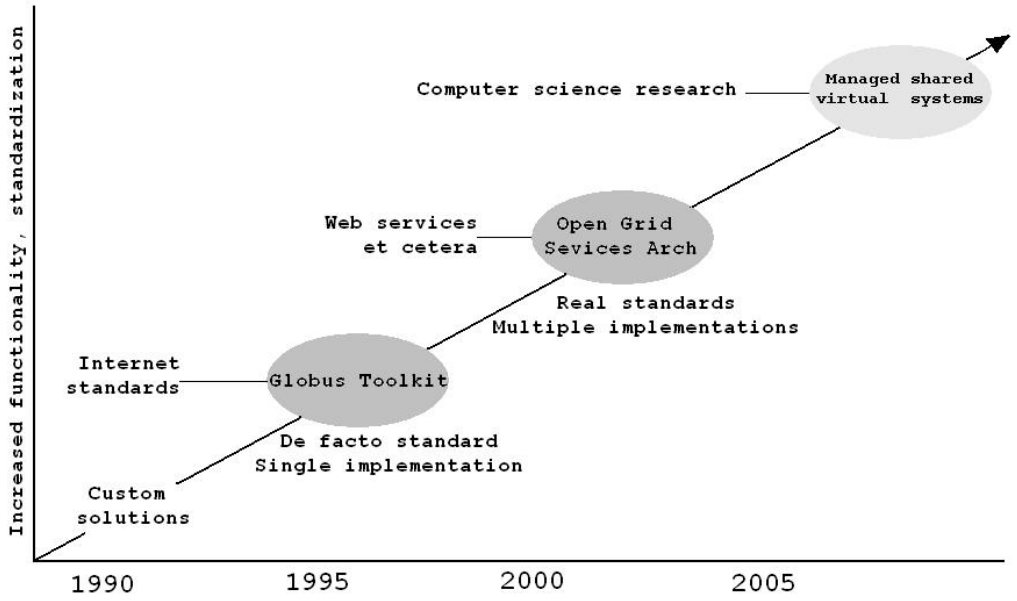
- koordinuje zdroje nepodléhající centrální správě
- používá standardní, otevřené a obecné protokoly a rozhraní
- poskytuje netriviální kvalitu služeb

2.4 Vývoj gridových technologií

Gridové technologie se začaly objevovat na počátku poslední dekády minulého století (obr. 2.1). Od této doby se objevily 4 fáze ve vývoji gridů:

Řešení na zákazku - jednalo se o první pokusy s gridy a metacomputingem. Důraz se kladl především na to, aby výsledné řešení „nějak fungovalo “. Aplikace byly stavěny přímo na protokolech Internetu s omezenou funkčností z pohledu bezpečnosti, rozšířitelnosti a robustnosti. Nekladl se příliš důraz na interoperabilitu výsledných řešení.

Globus Toolkit - Práce na Globus toolkitu (GT) začala v roce 1997 a od té doby se stal GT de facto standardem pro oblast gridových technlogií. Více informací o GT je uvedeno v kapitole 3 na straně 20.



Obrázek 2.1: Vývoj gridových technologií

OGSA - Open Grid Sevices Architecture - vrstevnatá architektura, která vznikla v roce 2002. Standard, který popisuje vznik, údržbu a aplikaci kompletních služeb podporovaných VO, viz. kapitola 2.8 na straně 15.

Spravované a sdílené virtuální systémy - budoucnost gridových technoligií umožňující větší stupeň virtualizace, dokonalejší interakci gridových systémů a bohatší sdílení zdrojů.

2.5 Grid vs. cluster

Z pohledu granularity výpočetních problémů existují dva hlavní typy: výpočty s *malou* a *velkou* granularitou.

U výpočtů s malou granularitou je každý podproblém vysoce závislý na výsledku dalších podproblémů. Z tohoto důvodu je výpočet takovýchto úloh v gridovém prostředí velmi složitý a tyto úlohy se počítají na paralelních superpočítačích či velmi pevně vázaných clusterech s procesory propojenými extrémně rychlou sítí.

Naopak u výpočtů s velkou granularitou je každý podproblém nezávislý na ostatních, a proto je snadné výpočet realizovat v gridovém prostředí. Tyto úlohy je vhodné řešit

pomocí sítě volně vázaných počítačů.

Reálné výpočty jsou kombinací obou typů úloh. Výpočet lze rozdělit na úlohy, které na sobě nejsou moc závislé, a paralelně je spočítat na několika clusterech na gridu. Gridy a clustery jsou primárně určeny pro odlišné typy úloh, ale jejich kombinace je nejvhodnější pro řešení komplexních úloh. Jako příklad by bylo možno uvést modelování podnebí Země, kdy se na jednotlivých clusterech modeluje pouze část Země a celá simulace probíhá v gridovém prostředí mnohem rychleji.

Další nevýhodou clusterů je jejich složení. Protože jsou složeny v podstatě ze stejných počítačů, nejsou vhodné pro některé typy úloh. Naopak u gridů lze využít specifických vlastností zapojených počítačů a například vektorové části úlohy je možné počítat na vektorových superpočítačích v gridu a skalární části na počítačích skalárních.

2.6 Organizace zabývající se gridy

Organizace zabývající se vývojem a implementací gridů lze rozdělit don čtyř hlavních kategorií.

2.6.1 Vývoj gridových standardů

Do této kategorie patří organizace zabývající se o zdokonalováním gridového standardizačního procesu. Mezi nejvýznamnější představitele patří *Global Grid Forum* (GGF)[14]. Tato organizace se stará o koordinovaný vývoj gridových technologií a spolupracuje s organizacemi jako například OASIS, W3C, IETF, DMTF a další... Mezi hlavní cíle GGF patří:

- Vytvoření otevřeného procesu vývoje gridových specifikací a jejich vývoj
- Správa a kontrola verzí dokumentů a specifikací zabývající se gridy
- Rozvoj spolupráce mezi lidmi zabývajícími se výzkumem gridů a uživateli
- Vzdělávání celé gridové komunity a šíření informací o gridech mezi další uživatele

2.6.2 Vývoj grid toolkitů, frameworků a middleware

Tato kategorie obsahuje mnoho projektů. Mezi nejznámější patří bezesporu *Globus* (více informací viz kapitola 3 na straně 20). Zde je uveden malý přehled ostatních hlavních projektů:

Legion - middleware projekt původem z University of Virginia[1]. Jedná se o vývoj objektově orientovaného softwaru pro gridové aplikace. Cílem projektu je podporovat zásady při vývoji softwaru pro distribuované systémy poskytováním standardních objektů pro procesory, datové systémy, souborové systémy, atd... Vývoj aplikací založených na Legionu je postaven na těchto standardních objektech. Více informací viz. [15].

Condor - jedná se o nástroj k využití kapacity nečinných pracovních stanic pro výpočetní úlohy. Je vhodný pro úlohy s velkou granularitou, tj. na sobě navzájem nezávislé. Condor se také používá pro správu clusterů v součinnosti s jinými gridovými technologiemi. Pro více informací viz. [16].

Condor-G - založený na systému Condor. Používá se hlavně pro správu služeb gridových technologií, například v součinnosti s projektem Globus (kapitola 3 na straně 20). Jde o kombinaci protokolů správy zdrojů Globusu (GRAM, Index Services) a metod správy zdrojů Condoru (obr. 2.2). Pro více informací o CONDORu viz. [16].

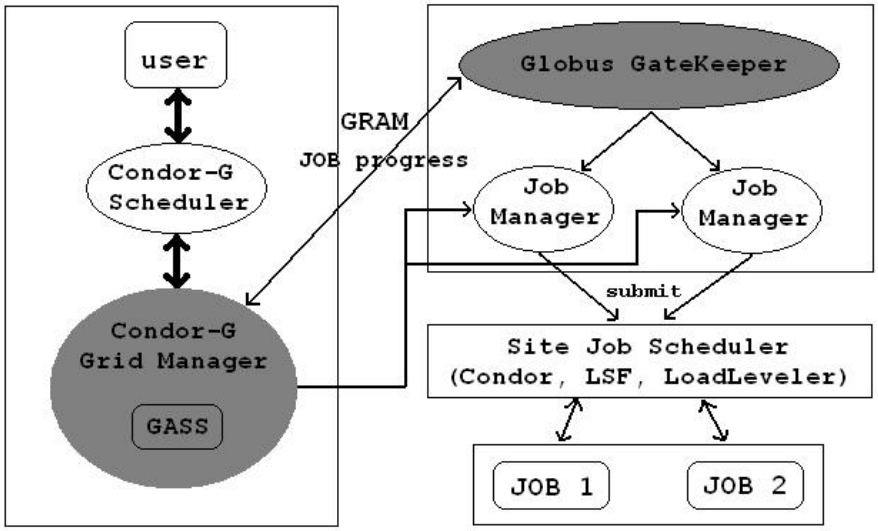
Mezi další významné projekty patří **Nimrod**[17], **UNICORE** (UNiform Interface to COmputer REsource)[18] a **NSF Middleware Initiative (NMI)**[19].

2.6.3 Implementace grid technologií ve společnostech

Zde je přehled nejvýznamnějších projektů zabývajících se implementací gridových technologií.

DOE:Science Grid - projekt vedený *United States Department of Energy*. Cílem je integrace superpočítačových center DOE do jednoho gridu, který by sloužil k vědeckým účelům[20].

[1] http://www.virginia.edu



Obrázek 2.2: Condor a Globus

EU:EUROGRID Project - projekt z grantu Evropské komise. Podílí se na něm 11 partnerů a 6 zemí EU s cílem vytvořit mezinárodní síť mezi superpočítačovými centry EU a zároveň s cílem vytvořit software pro tuto síť, která by ukázala přínosy gridových technologií a sloužila by k jejich dalšímu rozvoji[21]. Součástí projektu jsou projekty *BioGrid* (biomolekulární simulace), *MetroGrid* (výzkum a předpověď počasí), *Computer-Aided Engineering (CAE) Grid* (průmyslové CAE aplikace z automobilového a leteckého průmyslu) a *High Performance Center (HPC) Research Grid*.

EU:Data Grid Project - projekt sponzorovaný EU s účelem zpřístupnění geograficky rozmístěných superpočítačových center[22]. Tento grid slouží pro aplikace zpracování satelitních snímků Evropské vesmírné agentury, zpracování snímků z oblasti lékařství a biologie a pro potřeby CERNu.

EGEE - Enabling Grids for E-science in Europe[23]. Tohoto projektu se účastní i ČR v zastoupení *CESNETu*. Hlavním cílem je vybudovat produkční gridovou infrastrukturu na celoevropské úrovni. Projekt navazuje na projekt *Data Grid* a klade důraz na manipulaci s velkými objemy dat, která vzniknou po uvedení *LHC* (Velký urychlovač částic) v CERNu do provozu. Poté bude nutnost uložit a zpracovat přibližně **12 PB** dat (20 milionu CD) za rok.

K dalším významným projektům lze zařadit například **TeraGrid**[24], **NASA Information Power Grid (IPG)**[25]. V současné době probíhají jenom v zemích EU desítky gridových projektů.

2.6.4 Implementace grid technologií v komerční sféře

Vývoj gridů se již dostal do takové fáze, že je možné je nasadit v komerční sféře. Klíčové strategie pro jejich aplikaci je úplné využití výpočetního výkonu, virtualizace zdrojů a výpočty na vyžádání (on-demand computing). Jednou z důležitých věcí při vývoji gridů pro komerční nasazení je standardizace gridových technologií (viz OGSA - kapitola 15). Jako první se o implementaci gridů pokoušejí firmy *IBM* (Bussines On Demand solutions), *HP* (Utility computing and Data centers), *Sun Microsystems* a také *Microsoft*. Dá se očekávat, že aplikace založené na gridech budou v budoucnu hrát důležitou roli v IT systémech všech rozsáhlejších společností.

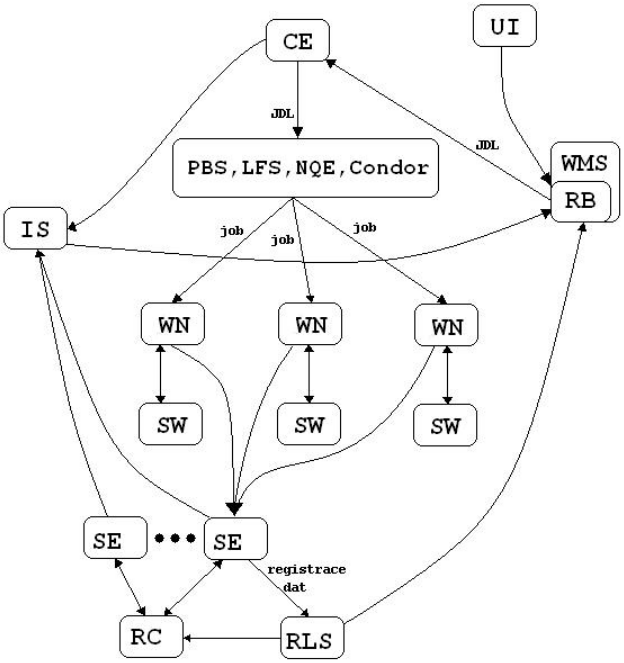
Již dnes se každý může zapojit do projektu *SETI@home*[26]. Cílem je analýza radio signálů z vesmíru za účelem nalezení umělého signálu. Je třeba analyzovat ohromné množství dat, a proto každý účastník dostane velmi malou část, kterou zpracuje, a výsledek pošle zpět. Při účasti velkého počtu uživatelů se dosáhne ohromného výpočetního výkonu. Tento projekt není grid podle všech vlastností gridů, ale je postaven na podobném principu. Dalším podobným projektem je *United Devices Cancer Research Project*[27], který analyzuje interakci mezi různými chemikáliemi a tímto způsobem se snaží nalézt látky, které ovlivňují rakovinu. Dva výše uvedené projekty byly vytvořeny společností *United Devices*[2], která se rovněž zabývá nasazením gridů v komerční praxi.

2.7 Struktura gridu

V této části je popsána struktura gridu. Následující popis se vztahuje k obrázku 2.3, kde je zobrazena struktura gridu, který vznikl jako výsledek projektu EGEE (kapitola 2.6.3 na straně 11). Tato struktura je pouze jeden z příkladů, jak může vypadat architektura gridu. Další příklad je uveden v kapitole 3 o Globus Toolkitu na straně 20.

UI - User Interface

[2] http://www.ud.com



Obrázek 2.3: Struktura gridu

- zprostředkovává uživatelům „přístup“do gridu
- tvoří rozhraní ke službám gridu
- umožňuje zadávat úlohy a sledovat jejich stav, zjišťovat informace o volných zdrojích, spravovat uživatelská data atd. . .

CE - Computing Element

- tvoří frontend pro konkrétní skupinu výpočetních uzlů
- přijímá joby určené pro konkrétní cluster/farmu
- poskytuje detailní informace o výpočetní kapacitě a nainstalovaném softwaru
- přijaté výpočetní úlohy předává lokálnímu dávkovému plánovacímu systému (např. Condor), který pošle úlohy ke zpracování na výpočetní prvky (WNs)

SE - Storage Element

- tvoří jednotné rozhraní k ukládání dat uživatelů gridu a umožňuje přístup k jednotlivým souborům

- soubory je možné replikovat a přistupovat k „nejbližší“replice
- každý registrovaný soubor má svou identifikaci v gridu, přes kterou se k němu přistupuje
- znalost přesného umístění souboru není nezbytná

WNs - Working Nodes

- tvoří vlastní výpočetní prvky - slouží ke zpracování jednotlivých úloh
- musí mít přístup k aplikačnímu softwaru
- musí mít nainstalován SW pro I/O dat na SE(grid-ftp)
- jediné nemusí být dosažitelné z „vnějšku“, ale pouze z CE

RC - Replica Catalog

RLS - Replica Location Server

- spravuje informace o replikách jednotlivých souborů
- slouží k administraci replikovaných souborů a výběru vhodné repliky

IS - Information Service

- informace o aktuálním stavu elementů gridu (CE, SE, ...)
- monitorování aktuálního stavu konkrétní úlohy

RB - Resource Broker

- plánovač, který vybírá optimální zdroje podle požadavků úlohy
- řídí rozdělování výpočetních úloh na jednotlivé CE, rozesílání JDL (Job Description Language) a samotných dat (pokud nejsou dostupná přes SE)
- rozhodování provádí na základě informací z IS
- je součástí sytému správy zatížení zdrojů (WMS) zajišťujícího vlastní distribuované plánování, správu zdrojů a optimalizaci jejich využívání

Celý systém funguje takto: Uživatel se přihlásí na UI a vytvoří si dočasný certifikát, který ho autentizuje při každé zabezpečené operaci a který má omezenou životnost. Uživatel zadá výpočetní úlohu (stav úlohy SUBMITTED). RB pomocí informací z IS najde nejvhodnější CE (stav WAIT). RB připraví úlohu na spuštění tím, že kolem něj vytvoří wrapper skript, který je spolu s ostatními parametry zaslán na CE (stav READY). CE přijme požadavek a zašle úlohu pro zpracování do lokálního dávkového systému (např. Condor)(stav SCHEDULED). Úloha je spuštěna na dostupných WNs, uživatelská data jsou zkopírována z RB na WN (stav RUNNING). Úloha může vyprodukovat nová výstupní data, která mohou být dána k dispozici ostatním uživatelům gridu. Pokud úloha úspěšně skončí, její výstup se zkopíruje zpět na RB (stav DONE). V případě, že se farma, kde job běží, stane nedostupnou, úloha je automaticky spuštěna jinde (rozhodne RB). Uživatel může spuštění zrušit (stav ABORTED).

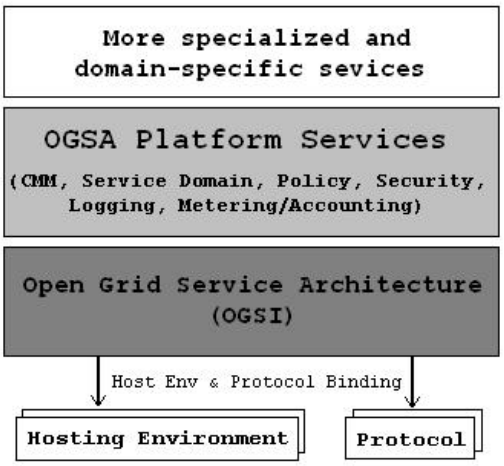
2.8 OGSA

Princip gridů je postaven na sdílení zdrojů a služeb, které se musejí nějakým způsobem vytvořit, spravovat a aplikovat. Mohou být vytvořeny jednou či více institucemi a poskytovat celou řadu funkcionalit. Z důvodu jejich různorodosti mohou být implementovány pro každou VO jinak. Tato situace se přímo nabízí pro to, aby se vytvořila jednotná specifikace pro vytváření, správu a aplikaci služeb udržovaných virtuálními organizacemi. Proto vznikla OGSA - **Open Grid Service Architecture**.

OGSA je vrstevnatá architektura s jasnou specifikací funkčnosti všech vrstev (viz obrázek 2.4). Platí, že výše postavené vrstvy v modelu využívají služeb spodních vrstev. OGSA určuje, co jsou to gridové služby, ale nezabývá se žádnými detailními ani technickými specifikacemi.

OGSA se snaží obsáhnout následujících cílů:

- usnadnit správu distribuovaných zdrojů skrze heterogenní platformy
- poskytovat bezproblémové doručení QoS
- publikovat otevřené standardy rozhraní a zpráv
- stavět na specifikacích OGSI a definovat architekturu a standardy pro základní gridové služby



Obrázek 2.4: Architektura gridů

Seznam základních služeb OGSA:

- Common Management Model (CMM)
- Service domains
- Přístup k distribuovaným datům a jejich replikace
- Policy
- Bezpečnost
- Provisioning and resource management
- Účtování
- Společné distribuované logování
- Monitorování
- Plánování

Více informací o OGSA lze nalézt v [5], [2], [14] nebo [6].

2.9 OGSI

OGSI (Open Grid Services Infrastructure) je základní komponenta OGSA architektury (viz kapitola 2.8, obrázek 2.4). Je to pokus o standardizaci infrastruktury gridového SW založený na standardech webových služeb za účelem dosáhnutí co největší interoperability mezi OGSA komponenty[5]. Jinými slovy, každá gridová služba je vlastně webová služba, která splňuje určitá pravidla daná OGSI. OGSI specifikace se stará zejména o pojmenovávání instancí, o společné rozhraní a chování všech gridových služeb a specifikaci dalších rozhraní. OGSI se tedy na rozdíl od OGSA zabývá formálními a technickými specifikacemi gridových služeb. Cílem OGSI není určit, jak mají být konkrétní služby implementovány, ale jak mají komunikovat se svým okolím, aby bylo možné dané služby implementovat na jakékoliv platformě.

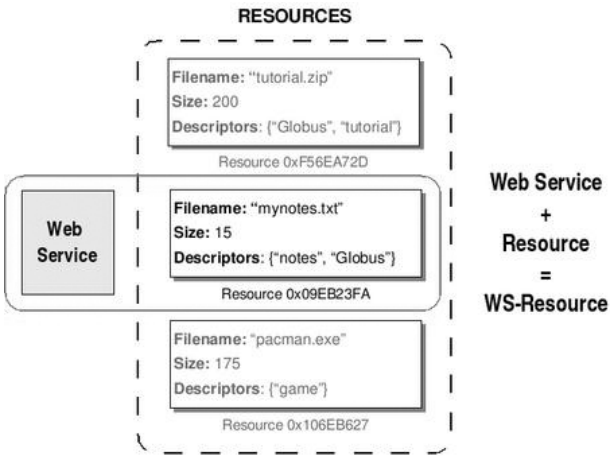
Specifikace OGSI je úzce spojena s webovými službami. Ty jsou obecně bezestavové a komunikace s klientem probíhá také bezestavově. Takováto komunikace je pro gridy nevyhovující, protože gridové služby jsou obecně déletrvající činnosti, u kterých je dobré znát stav a jiné údaje. Z tohoto důvodu má každá gridová služba stav, který je možné kdykoliv zjistit. OGSI se snaží zajistit, aby zprávy vyměňované mezi klientem a gridovou službou byly stavové a aby probíhaly pod jednotnou specifikací. Ke komunikaci se využívají XML zprávy a gridové služby (respektive jejich rozhraní) jsou popsány jazykem GWSDL (Grid Web Service Description Language), který vznikl rozšířením jazyka WSDL.

Více informací o OGSI lze nalézt v [5] nebo [14].

2.10 WSRF

V posledních letech se vývoj webových a gridových služeb ubíral jinými směry. Pro webové služby vznikla nová verze 2.0 jazyka WSDL. Gridové služby byly postaveny na specifikacích OGSA a OGSI. Toto se nakonec ukázalo jako nežádoucí a proto vznikl společný standard pro webové a gridové služby - *Web Services Resource Framework* (WSRF). Na tomto standardu je založena nová verze Globus Toolkitu 4.

WSRF specifikuje způsob, jak vytvořit z bezestavových webových služeb služby stavové. V OGSA a OGSI bylo tohoto výsledku dosaženo přidáním dat (*Service Data Element*) ke každé gridové službě. SDE obsahuje všechny stavové informace o gridové službě.



Obrázek 2.5: WS-Resource

WSRF podobně specifikuje entity zvané *resource*, které jsou od služby také odděleny a obsahují informace, které byly dříve uloženy v SDE. Každý resource je identifikován klíčem a jedna služba může postupně využívat více takovýchto zdrojů. Klient se již nepřipojuje ke specifické službě, která obsahuje SDE, ale k obecné službě s daným klíčem zdroje. Z tohoto důvodu se zavádí pojem *WS-Resource*, který specifikuje adresu služby+zdroje (obrázek 2.5). Mechanismus práce se zdroji ve WSRF je deailně vysvětlen v kapitole 4.2 na straně 30.

WSRF obsahuje těchto 7 specifkací, které se starají o správu WS-Resource:

WS-ResourceProperties - specifikace definice a přístupu k atributům zdroje

WS-ResourceLifetime - poskytuje základní mechanismy, které se starají o správu životního cyklu zdrojů služby

WS-ServiceGroup - specifikuje jakým způsobem lze ze zdrojů a služeb vytvářet skupiny a jak tyto skupiny spravovat

WS-BaseFaults - poskytuje způsob, jak pracovat s chybami, které mohou nastat při vytváření WS-Resource

WS-Notification - umožňuje automatické posílání zpáv jako reakci na určté události, které mohou nastat například při změně atributů zdroje

WS-Addressing - poskytuje mechanismus, jak adresovat webové služby v souvislosti se zdroji

2.11 Další informace o gridech

Více informací o gridech je obsaženo v knize **The Grid2: Blueprint for a New Computing Infrastructure** [2] autorů Iana Fostera a Carla Kesselmana, která se zabývá především obecnými principy z oblasti gridů a uvádí přes 700 odkazů na další publikace z této oblasti.

Kapitola 3

Globus Toolkit

Globus Toolkit je jedna z nejlépe vytvořených implementací OGSI standardů (nově také implementace WSRF) a používá se jako výchozí bod pro mnoho gridových projektů. V následujících částech této kapitoly jsou obsaženy informace o historii a struktuře Globus Toolkitu.

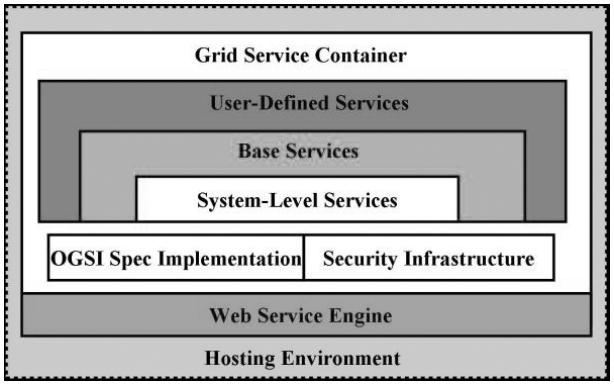
3.1 Historie Globus Toolkitu

Globus Toolkit (GT)[13] je open source software toolkit vyvíjený komunitou *Globus Alliance*[1]. Tato komunita se stará o vývoj základních technologií z oblasti gridů. Historie GT se datuje do roku 1994, kdy Rick Stevens, ředitel divize matematiky a počítačových věd v *Argonne National Laboratory a Tom DeFanti* z *University of Illinois* vytvořili dočasné spojení mezi 11 vysokorychlostními sítěmi a tím vytvořili národní grid nazvaný *I-WAY*. Experiment byl proveden pro účely *Superpočítačové konference* v roce 1995. Pro tento grid vyvinul tým vedený Ianem Fosterem nové protokoly, které umožnily uživatelům gridu I-WAY spouštět aplikace na druhé straně země. Po tomto úspěchu byl agenturou DARPA (Defense Advanced Research Projects Agency)[2] financován další vývoj experimentu a to v roce 1997 vedlo k první verzi Globus Toolkitu, který se brzy používal na 80 místech po celém světě. Do současné doby získal projekt GT mnoho ocenění a byl gridovou komunitou přijat jako de facto standard na poli gridových technologií.

Verze 2 byla uvedena v roce 2002 a byla postavena na technologii LDAP. Verze 3 (GT3)

[1] http://www.globus.org/alliance

[2] DARPA byla podobně i u vzniku Internetu v roce 1969



Obrázek 3.1: Struktura GT3

(viz kapitola 3.2 na straně 21) vychází z OGSI specifikací a v současné době se ve většině systémů používá právě tato verze. Na jaře roku 2005 byla vydána zatím poslední verze Globus Toolkitu - GT4. GT4 (kapitola 3.3 na straně 27) je založen na WSRF a oproti verzi 3 bylo provedeno několik změn hlavně v oblasti práce se zdroji a službami.

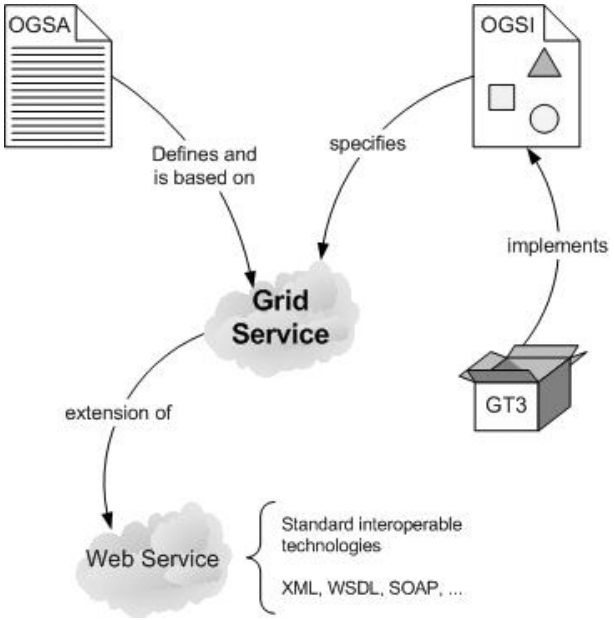
3.2 GT3

Na obrázku 3.1 je zobrazena struktura jádra GT3. Jádro obsahuje klíčové komponenty, které jsou nutné pro vytvoření gridových služeb. Jak je vidět z obrázku, architekturu GT3 lze rozdělit do čtyř oblastí:

- Základní služby - Base services
- Uživatelem definované služby - User-defined services
- Jádro GT3
- Kontejner gridových služeb

Základní služby jsou postaveny na jádře GT3 a z toho důvodu využívají služby, které jádro poskytuje. Více informací o těchto službách je uvedeno v kapitole 3.2.2 na straně 23.

Uživatelem definované služby jsou služby na aplikační úrovni, které využívají implementace a bezpečnostní infrastruktury OGSI specifikace. Tyto služby mohou běžet v součinnosti s jinými službami. Jako příklad lze uvést správce alokace zdrojů a monitorovací služby.



Obrázek 3.2: Vztah mezi OGSA, OGSI a GT3

Na obrázku 3.2 je zobrazen vztah mezi OGSA, OGSI a GT3.

3.2.1 Jádro GT3

Jádro Globus Toolitu 3 se skládá ze tří hlavních částí (obr. 3.1):

- Referenční implementace OGSI - OGSI Reference implementation
- Bezpečnostní infrastruktura - Security infrastructure
- Služby na systémové úrovni - System-level services

Část jádra nazvaná *Referenční implementace OGSI* je implementace všech rozhraní definovaných OGSI a poskytuje API a další nástroje, které ulehčují vývoj služeb kompatibilních s OGSI. Standardní rozhraní OGSI jsou například následující: GridService, Factory, Notification(Source/ Sink/ Subscription), HandleResolver, ServiceGroup(Entry/ Registration).

Bezpečnostní infrastruktura se stará o zabezpečení vyměňovaných zpráv, jejich autentifikaci a autorizaci. To je docíleno pomocí GSI (kapitola 3.2.2) a PKI standardů a probíhá na úrovni zpráv (SOAP úrovni) nebo na transportní úrovni (používá se protokol *httpg* za-

ložený na HTTP a GSI, podobný protokolu *https*)[3]. U zabezpečení na úrovni zpráv se využívají dva typy bezpečnostních mechanismů: *GSI Secure Conversation* a *GSI XML Signature*. Při využití GSI Secure Conversation klient nejprve naváže bezpečné spojení se službou pomocí *Secure Conversation Service*. Poté je toto spojení používáno pro kódování zpráv. Při použití GSI XML Signature je použito X.509 certifikátu pro zakódování zprávy.

Dva výše uvedené bloky jádra GT3 neposkytují žádné služby, ale slouží jako nezbytný základ pro vytváření služeb. O to se stará část jádra GT3 nazvaná *System level services*, která využívá všech možností poskytnutých spodními dvěma vrstvami. Tato část poskytuje na sobě nezávisle pracující základní služby v následujících oblastech: *Admin* (správa kontejneru), *Logging* (modifikace a monitorování logů) a *Management* (monitorování a správa služeb kontejneru).

Všechny výše uvedené části GT3 mezi sebou spolupracují v abstraktním OGSI run-time prostředí, které se nazývá *Kontejner gridových služeb* (Grid Service Container). Jeho účelem je vytvořit pomyslný štít mezi aplikací a specifickými run-time vlastnostmi prostředí, ve kterém je aplikace spuštěna. Kontejner se zároveň stará o správu životního cyklu služeb, o doručení zpráv jednotlivým instancím služeb a zároveň spolupracuje s XML zprávami prostřednictvím *Web Service Engine*.

Grid Service Container a Web Service Engine jsou umístěny v *Hosting Environment*, což je webový server, který se stará například o správné zpracování HTTP zpráv.

Více informací o jádře GT3 lze nalézt v dokumentu [8] a na stránkách projektu Globus [13].

3.2.2 Základní služby GT3 - Security

GSI - *Grid Security Infrastructure*. Zabezpečení v GT3 je postaveno na principech kryptografie s veřejnými klíči (asymetrická kryptografie). Princip spočívá v existenci dvou klíčů a funguje tak, že data šifrovaná jedním z klíčů lze v rozumném čase dešifrovat pouze se znalostí druhého z dvojice klíčů a naopak. Jeden z nich, takzvaný privátní klíč je s maximální bezpečností ukrýván majitelem (čipové karty, flash disk v trezoru, atd.), zatímco druhý klíč je zveřejněn. Protože je veřejný klíč obecně znám všem, nelze zprávu zašifrovanou s použitím privátního klíče považovat za zašifrovanou v plném smyslu slova (důvěrnou), ale pouze za podepsanou. Proto se v praxi při zakódování používá nejdříve soukromý klíč

[3] U verzí novějších než GT3 se bude postupně upouštět od zabezpečení na transportní úrovni

odesílatele a poté veřejný klíč adresáta. Dekódování probíhá naopak.

Každý, kdo se chce nějak zapojit do fungujícího gridu (uživatel nebo služba), musí mít *certifikát*. Ten je zakódován podle standardu X.509 a obsahuje jméno subjektu, veřejný klíč subjektu, identitu *certifikační autority* (CA), která podepsala klíč, aby zaručila, že tento veřejný klíč patří tomuto subjektu a jako poslední obsahuje podpis CA. Při ověřování certifikátu subjektu musí existovat nějaká cesta, jak ověřit, jestli daná CA je důvěryhodná. GSI používá pro vzájemnou autentifikaci vrstvu TLS (dříve známou jako SSL).

Poté, co byly úspěšně ověřeny identity subjektů, další komunikace již probíhá bez využití GSI. Lze si dodatečně vyžádat zabezpečení komunikace vygenerováním klíče pouze pro tuto komunikaci. Defaultně probíhá komunikace se zajištěnou integritou. To znamená, že lze vzájemnou komunikaci dvou subjektů „poslouchat“, ale přenášené zprávy nelze měnit.

V systémech s GT se předpokládá, že soukromé klíče jsou uloženy v souborech na disku. Aby se zabránilo zneužití těchto klíčů, je každý soubor s klíčem zabezpečen heslem, bez jehož znalosti se ke klíči nelze dostat. V praxi by se často stávalo, že by byl uživatel nucen zadávat pokaždé heslo pro přístup ke svému soukromému klíči, a proto se zavádí takzvaná *proxy*. Ta má také svůj soukromý a veřejný klíč podepsaný uživatelem a platnost klíče je časově omezena. Proxy se využívá k dočasnému zastoupení uživatele při autentifikaci s jinými subjekty. CA podepisuje klíč uživatele, uživatel podepisuje klíč proxy, a proto lze dokázat autenticitu proxy klíče.

CAS - *Community Authorization Service*. Jedná se o povolení přístupu ke zdrojům podle komunity. Komunita je seznam uživatelů a má svoje GSI pověření, které komunitu reprezentuje jako celek skrze CAS server, běžící v komunitě. Poskytovatelé zdrojů povolují přístup ke zdrojům dle práv pro jednotlivé komunity. Pokud chce uživatel získat přístup k nějakému zdroji, tak si nejprve musí vyžádat pověření od příslušného CAS serveru, že patří k dané komunitě, a poté se s tímto pověřením snaží normálními mechanismy získat povolení pro přístup ke zdroji.

3.2.3 Základní služby GT3 - Data Management

GridFTP - Jde o vysoce výkonný, zabezpečený a spolehlivý protokol pro přenos dat založený na *FTP* protokolu. Tento protokol vznikl z důvodu neexistence dostatečně vý-

konného protokolu, který by byl schopen zajistit všechny následující požadavky:

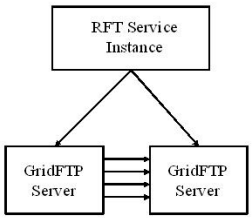
- souvislé přenosy velkých objemů dat
- základ v průmyslových standardech
- bezpečnost: autentifikace, autorizace, integrita a privátnost dat
- rychlost: možnost paralelismu s nízkými nadbytečnými náklady na přenos
- robustnost: řešení selhání systémů
- umožnit kontrolu přenosu dat „třetí stranou“
- integrované nástroje pro změnu parametrů již probíhajících přenosů (např. změna velikosti bufferu TCP)
- snadná rozšiřitelnost

Jako nevyhovující protokoly pro přenos dat v gridech se ukázaly tyto protokoly: *FTP*, *HTTP* (+ jejich verze), *HPSS* (firma IBM), *DPSS* (firma LBNL) a *SRB* (firma SDSC). U posledních 3 protokolů bylo hlavní překážkou jejich vlastnictví firmami. Nakonec padlo rozhodnutí předělat FTP protokol, přičemž největší předností FTP oproti HTTP je oddělení řídícího a datového kanálu. Oproti výše uvedeným vlastnostem umožňuje GridFTP navíc přenos pouze částí souborů, přenos dat přes několik serverů najednou s cílem zvýšení propustnosti přenosu a integrovanou bezpečnost s použitím *GSI* a *Kerberos*.

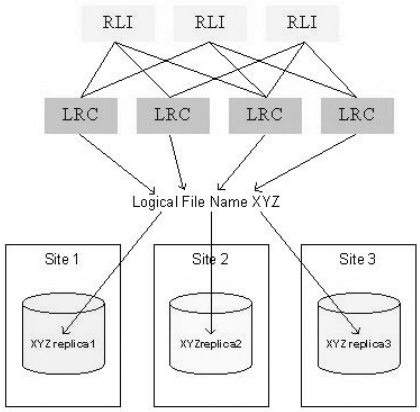
RFT - *Reliable File Transfer* - služba implementována podle OGSI specifikace. Slouží ke spolehlivému přenosu dat mezi dvěma *GridFTP* servery (obrázek 3.3). Rozdíl mezi *RFT* a *GridFTP* je v tom, že GridFTP není OGSI kompatibilní. RFT využívá existující GridFTP protokoly a nástroje pro přenos dat a také umožňuje měnit vlastnosti přenosu (TCP buffer, paralelismus atd...)

Více informací o RFT je uvedeno v [9] nebo [10].

RLS - Replica Location Service je služba umožňující mapování logických jmen souborů (LFN) na fyzická jména souborů (PFN). RLS byla vyvinuta ve spolupráci s DataGrid[22] projektem a nahradila *replica catalog* v GT2.x. Replikace dat snižuje přístupovou dobu



Obrázek 3.3: Reliable File Transfer



Obrázek 3.4: RLS

k datům, robustnost aplikace a celkově zvyšuje výkon distribuovaných aplikací. RLS samozřejmě nefuguje samostatně, ale v součinnosti s ostatními komponentami gridové architektury (viz kapitola 2.7 na straně 12).

RLS je jednoduchý distribuovaný registr, který obsahuje informace o umístění dat a jejich kopií (replik) v gridu (obrázek 3.4). Distribuovaný proto, protože může najednou existovat na několika serverech umístěných v různých částech gridu. To má výhodu oproti centralizovanému katalogu informací ve zvýšeném výkonu celého systému a takovýto systém je méně náchylný na poruchy. RLS může být provozován i v nedistribuované verzi jako centrální server. Při vytvoření souboru je nutné soubor zaregistrovat a pokud uživatel nebo služba nějaký soubor potřebují, zeptají se RLC (viz dále).

Konzistentní lokální informace o příslušném mapování jmen (LFN - PFN) jsou uloženy v *Local Replica Catalog* (LRC). Zde se registrují nové soubory a zde se rovněž odpovídá na dotazy, kde je uložen daný soubor. Nejjednodušší implementace RLS se skládá z 1 RLC serveru. Při distribuované implementaci RLS existuje *Replica Location Index* (RLI) server,

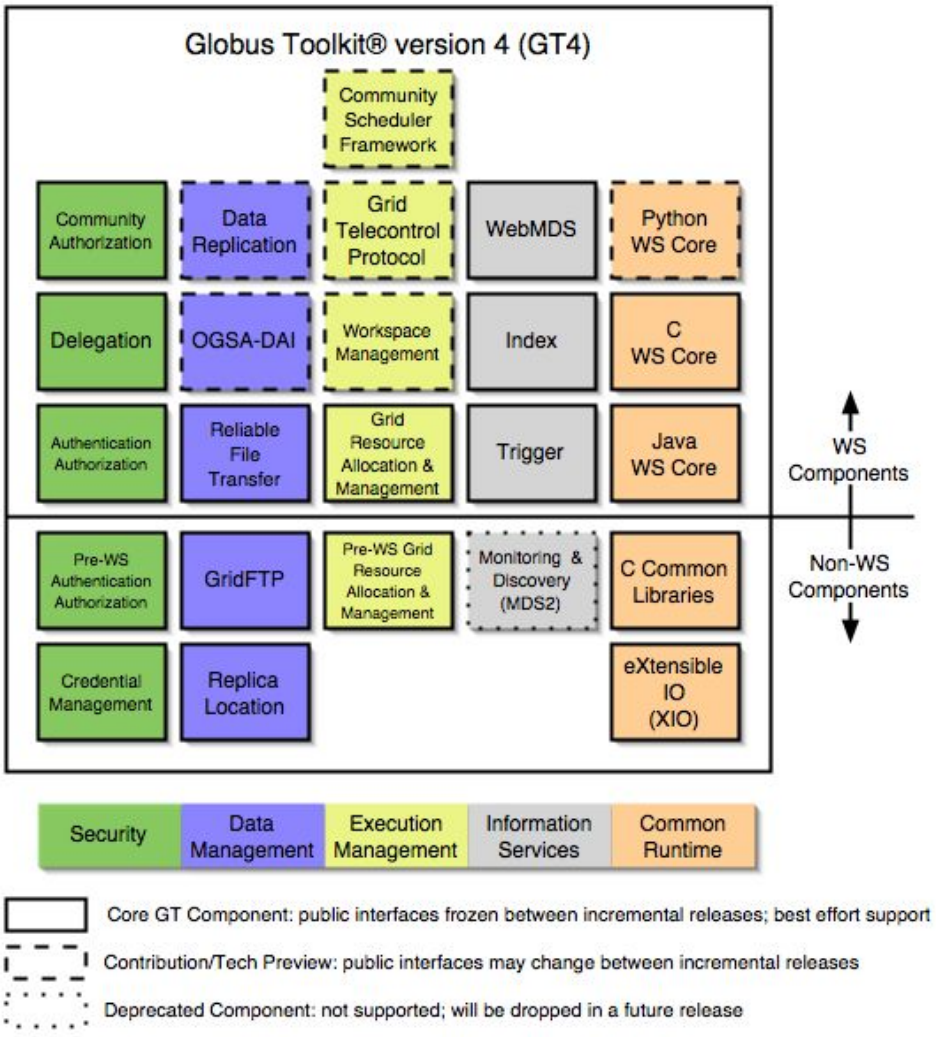
který obsahuje informace o logických jménech na několika LRC. Při dotazu, kde je nějaký soubor umístěn, je nejprve dotázán RLI server, který vrátí seznam LRC serverů, jež obsahují informace o uložení dotazovaného souboru (obrázek 3.4).

XIO - vysoce optimalizovaná knihovna pro input/output operace, která poskytuje jednoduché a intuitivní API všem gridovým IO protokolům. Globus XIO je navržena tak, aby urychlila vývoj a implementaci nových protokolů. Je snadno rozšiřitelná pomocí nových driverů a obsahuje timeouty pro IO operace.

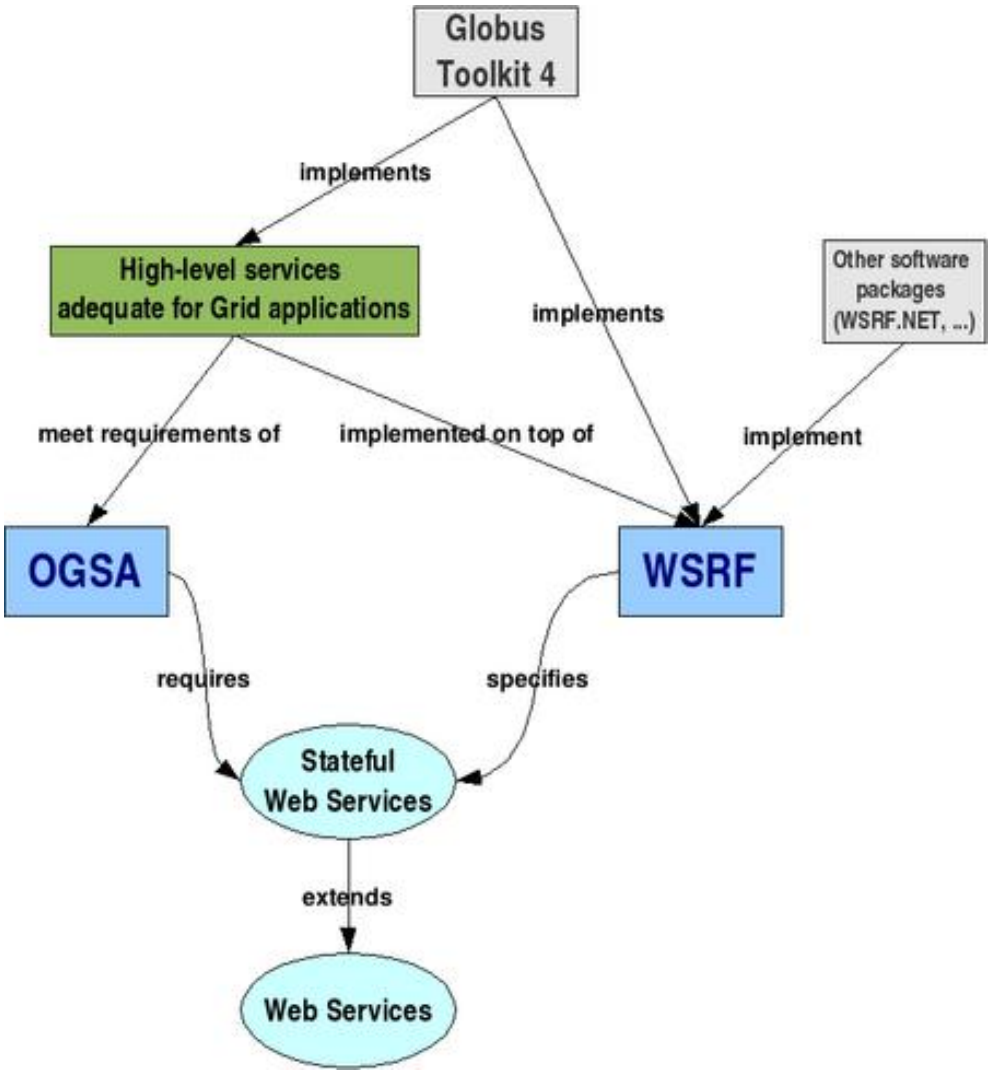
Více informací o GT3 a jeho částech lze nalézt v [13].

3.3 GT4

Na obrázku 3.5 je zobrazena struktura GT4, který se skládá z mnoha komponent, rozdělených do pěti oblastí. Komponenty ve spodní části obrázku nevycházejí ze specifikace WS (například GridFTP). Mnoho komponent z GT4 již bylo popsáno v předchozích částech této kapitoly. Hlavní změna oproti GT3 je v tom, že GT4 vychází z WSRF specifikací. Na obrázku 3.6 je znázorněn vztah mezi GT4, OGSA a WSRF. Více informací o vytváření služeb v GT4 je uvedeno v kapitole 4 na straně 30.



Obrázek 3.5: Struktura GT4



Obrázek 3.6: Vztah mezi GT4, OGSA a WSRF

Kapitola 4

Služby v GT4

4.1 Co je to služba

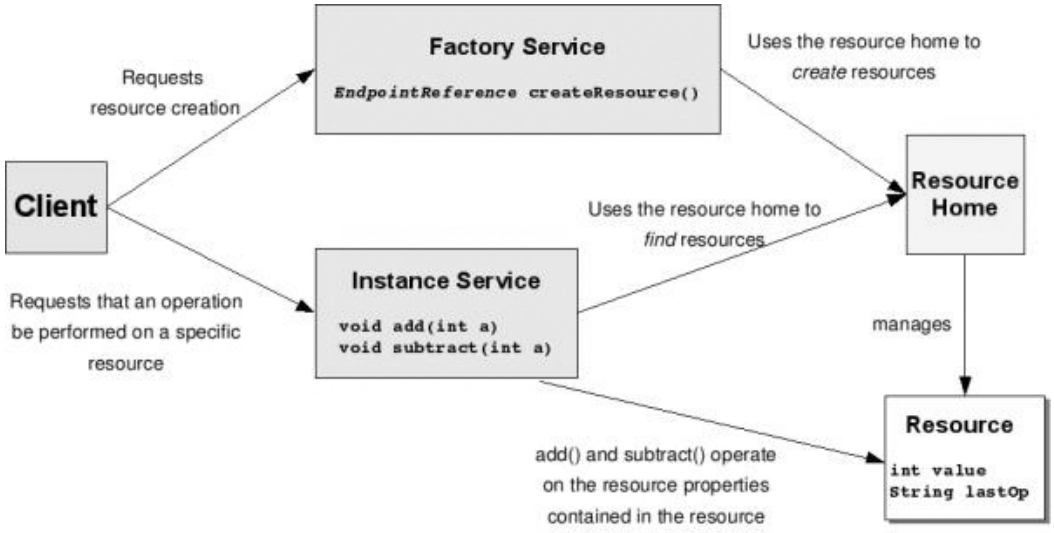
Každá gridová služba (GS) je založena na principu webových služeb (WS). Přenos a kódování přenášených zpráv mezi klientem a systémem s gridem je prováděno pomocí SOAP/HTTP. Instance gridových služeb jsou adresovatelné, částečně stavové a implementující jedno nebo více rozhraní nazvaných *portType*. Každá takováto instance je vytvořena příslušnou *Factory* a klient s ní může pracovat skrze *Endpoint Reference*, což je ukazatel na dvojici služba-zdroj, který vrací Factory. Pro popis veřejných rozhraní gridových služeb se používá jazyk *WSDL* (Web Services Description Language)[1].

Životní cyklus každé služby začíná vytvořením její instance při spuštění kontejneru, ve kterém je daná služba nainstalována. Poté se po připojení klienta k dané službě (respektive k factory služby) pouze vytvářejí a ruší zdroje, se kterými služba pracuje. Podobně služba zaniká po ukončení kontejneru.

4.2 Struktura služby v GT4

U gridových služeb je doporučeným přístupem pro práci se službami vzor *factory/instance*. Za tvorbu instancí zdrojů je odpovědna factory přes předem definovanou operaci. V GT4 se pomocí factory nevytvářejí instance služeb, ale pouze instance zdrojů (resource). Na obrázku 4.1 je zobrazena tato struktura.

[1] Více informací o WSDL lze nalézt na *http://www.w3.org/TR/wsdl*



Obrázek 4.1: Implementace služby v GT4

Klient zná při volání služby pouze adresu factory, která vytvoří zdroj. Každý zdroj má přiřazen unikátní klíč a tento klíč je vrácen factory, která ho vrátí jako *Endpoit Reference* (EPR) klientovi. V EPR je uložena dvojice služba/klíč, kterou klient použije pro volání služby, pracující se zdrojem, jež odpovídá hodnotě klíče z factory (obrázek 4.1).

Pro vytvoření zdroje použije factory *resource home*, který se stará o vytváření, správu a nalezení zdrojů. Resource home je také použit pro nalezení zdroje (podle klíče zdroje z EPR z factory) kdykoliv, když daná instance služby potřebuje provést nějakou operaci se zdrojem.

Celý postup vypadá následovně: Klient volá *factory* (pomocí jejího URI). Ta s využitím mechanismů, implementovaných v Globusu (konkrétně pomocí třídy *ResourceContext*), najde příslušný *resource home*, který vytvoří nový zdroj a přidá si ho do seznamu zdrojů. Tomuto zdroji je přiřazen jednoznačný identifikátor (nejčastěji číslo typu *int*). Tento identifikátor je přes resource home vrácen factory, která vytvoří *WS-Resource EPR* a vrátí jí klientovi. EPR obsahuje adresu služby, která umožňuje klientovi volat operace dané služby a klíč nově vytvořeného zdroje. Tento EPR použije klient pro nalezení příslušného *portType* již běžící služby. Pokud klient vyžaduje nějaké operace se zdrojem (prostřednictvím volané služby), pak služba použije informace o zdroji z EPR k jeho nalezení a poté provede příslušnou operaci přímo se zdrojem. Nalezení zdroje se děje opět prostřednictvím resource home.

Ke správnému přeložení, nainstalování a spuštění služby v kontejneru jsou potřeba tyto soubory:

- .wsdl - zvlášť soubory pro factory a pro instanci služby. Zde jsou popsána rozhraní (portType) tříd, se kterými komunikuje klient
- .java - implementace factory, služby, resource a resource home
- deploy-server.wsdd - tento soubor určuje způsob, jak budou služby zpřístupněny v kontejneru
- deploy-jni-config.xml - podle tohoto souboru kontejner pozná, jaký má použít resource a resource home v souvislosti s našimi službami

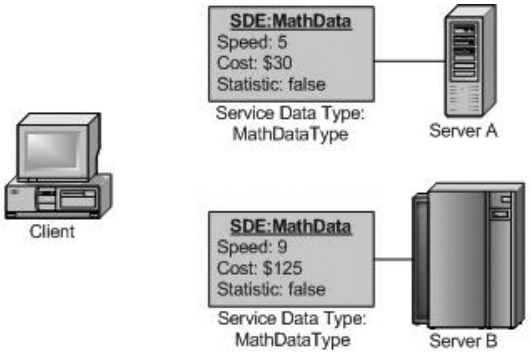
Ke kompletnímu seznamu souborů chybí soubory nutné pro překlad služby ze zdrojových textů. Ty budou uvedeny v kapitole 6 na straně 40 spolu s detailním vysvětlením obsahů výše uvedených souborů.

4.2.1 Rozdíl v implementaci služeb mezi GT3 a GT4

V GT3 může gridová služba obsahovat jeden nebo více *Service Data Elementů* (SDE). Každý takovýto SDE obsahuje strukturovaná data, která jsou vnitřně reprezentována pomocí XML a která slouží k popisu dané služby. Například na obrázku 4.2 je zobrazen příklad 2 instancí služby s SDE, která provádí nějakou matematickou operaci. SDE zde reprezentují předem dané vlastnosti služby. Klient nebo ostatní služby mohou díky znalosti dat z SDE upravit své chování k dané službě. SDE jsou popsána jazykem *GWSDL*. Každá služba v GT3 má unikátní *Grid Service Handle* (GSH), pomocí kterého lze se službou pracovat, a platí, že jedna služba může pracovat pouze s jednou instancí od několika typů SDE.

Naopak v GT4 neexistují SDE. Existují zde zdroje s atributy spravované pomocí resource home a jedna služba může při více volání klientem pracovat s více zdroji. To je dáno schématem WS-Resource specifikace WSRF.

Další rozdíl v implementaci služeb mezi GT3 a GT4 je v tom, že v GT3 se po zavolání factory vytvořila nová instance služby (společně s SDE) a klientovi byl vrácen ukazatel GSH na tuto službu. Klient poté pracoval s konkrétní instancí dané služby. V GT4 je instance služby vytvořena po startu kontejneru. Poté jsou voláním factory vytvářeny



Obrázek 4.2: Service Data Element

pouze nové zdroje a klientovi je vrácena *Endpoint Reference* na dvojici služba-zdroj (WS-Resource).

4.3 GRAM a MDS

GRAM je souhrnný název pro množinu komponent, které poskytují snadný přístup ke zdrojům a spouštění úloh ve vzdálených systémech a stal se v této oblasti základním stavebním kamenem pro mnoho současných projektů. S využitím GSI je GRAM vytvořen tak, aby poskytoval společné protokoly a API pro práci se systémovými zdroji. V současné době využívají lokální systémy pro přístup ke zdrojům několik mechanismů (plánovače, fronty, rezervace atd. . .). GRAM se snaží všechny tyto mechanismy skrýt před uživatelem tak, aby se uživatel musel naučit pouze jak používat GRAM a o ostatní věci by se nemusel starat. Ideu GRAMu lze přiblížit jako „přesýpací hodiny“, ve kterých je GRAM zúžená prostřední čast hodin, aplikace a služby vyšších úrovní jsou nahoře a nízkoúrovňové služby systému jsou dole.

GRAM neposkytuje funkce jako plánování, zprostředkování výpočetních kapacit a účtování. Tyto věci nechává na ostatních komponentách gridu.

Kapitola 5

Mobilita

V této kapitole je obsažena analýza mobilního přístupu ke gridům.

5.1 Mobilní přístup ke gridům

Tak jako se gridy neustále rozšiřují a obsahují více uzlů, tak současně s tím vzrůstá potřeba pro přístup ke gridům z téměř jakéhokoliv zařízení a místa. V současné době jsou gridy stavěny na aplikace, které mohou řešit téměř jakýkoliv výpočetně náročný úkol. Lze například simulovat chování zákazníků v obchodním domě, průběh záplav či zemětřesení.

Jako příklad lze uvést grid, který je určen pro simulaci začínajících záplav. Spuštění simulace lze provést s daty od meteorologické služby. Po určité době se simulace může stát nepřesnou vzhledem k neustále se měnícímu počasí a nedostatku aktuálních dat z terénu. Bylo by optimální měnit průběh simulace dodáváním aktuálních dat přímo do gridu, například z PDA zařízení. Současně s tím by bylo ideální mít na PDA neustále co možná nejpřesnější výsledky simulace pro určité území. Obdobný příklad by šel uvést u manažera velkého obchodního domu, který by měnil z letiště data simulace očekávané nákupní vlny v obchodě a ihned by byl schopen na základě výsledků simulace objednat nové zboží.

Využití gridů není pouze v jejich výpočetní kapacitě, ale v jakémkoliv sdílení zdrojů. Toho lze s úspěchem využít v různých e-learningových kurzech, kde lze sdílet materiály pro kurz. Nebo například nemocnice mohou sdílet informace o pacientech s lékaři a výjezdovými skupinami rychlých záchranných služeb. Další využití je také možné ve sdílení informací pro turisty v závislosti na jejich místě pobytu.

Mobilní klienti, kteří by takto přistupovali ke gridu, budou mít přirozeně různě vysokou výpočetní kapacitu a zobrazovací možnosti. Notebook bude schopen zobrazit výsledky simulace záplav s mnohem většími detaily a na větším území než například PDA. Naopak PDA nebo mobilnímu telefonu bude stačit pouze dílčí informace nebo SMS zpráva.

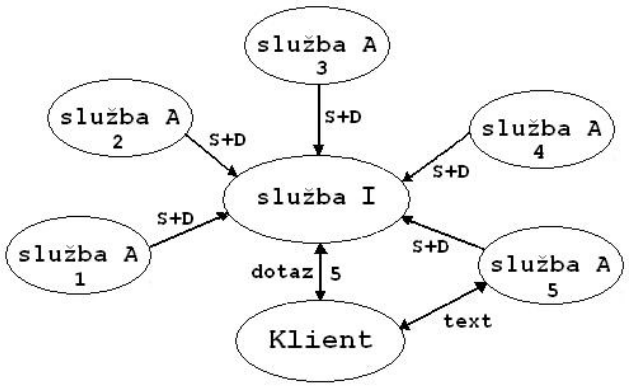
Tato zařízení se budou ke gridu připojovat přes spojení s rozdílnými vlastnostmi. Část se může připojit přes lokální sítě a část pouze bezdrátovými technologiemi. Někde bude rychlejšího připojení potřeba a někde bude mnohem více důležitá mobilita zařízení. Připojení budou mít také různou tarifikaci. Část bude platit za čas připojení, část za přenesená data a část bude mít například paušální platby bez omezení přenesených dat. Různé druhy připojení se budou dále také lišit spolehlivostí přenosů, zpožděním, formáty a aktuálností přenášených zpráv. Zařízení budou mít také různé operační sytémy.

Pokud chceme, aby se ke gridu mohli připojovat klienti výše popsanými způsoby, tak v gridu musí existovat mechanismus, který na základě dat od klienta rozhodne, jakou službu spustí či jakou formou komunikace bude pracovat dál s klientem. Aby mohl rozhodnout co nejoptimálněji pro klienta, tak od něj bude muset obdržet co nejvíce dat, na jejichž základě se bude rozhodovat. Následující část kapitoly se zabývá analýzou několika možností, jak by mohla vypadat interakce grid - mobilní klient.

5.2 Možnosti interakce klient-grid

Analýza je provedena na imaginární službě, která obsahuje dva typy dat, které popisují vlastnosti této služby. Prvním typem jsou statická data, která má služba od svého vytvoření a která se již dále nemění. Druhým typem jsou data dynamická. Tato data se mění v čase a popisují vlastnosti služby, jako je například aktuální zatížení serveru, rychlost přenosu dat po lince, atd.

Při rozhodování, kdo provede výběr optimální služby pro daného klienta, existují dvě možnosti. Za prvé může rozhodnout nějaká specializovaná služba v gridu, ke které se klient připojí a předá jí své požadavky. Za druhé může toto rozhodnutí učinit sám klient na základě atributů, které získá od služeb, mezi nimiž se bude rozhodovat.



Obrázek 5.1: Nalezení optimální služby jinou službou

5.2.1 Rozhoduje služba

Tento případ rozhodování je zobrazen na obrázku 5.1. Klient chce provést nějakou časově náročnou operaci se službou A, kterých je v systému několik. Jelikož klient o všech dostupných službách A neví, zeptá se služby I, aby mu našla nejvhodnější službu A. Služba I udržuje kontakt se všemi službami A a tudíž zná jejich vlastnosti. Poté podle nějakého algoritmu rozhodne, která služba je pro daného klienta nejvhodnější, a klientovi vrátí ukazatel na tuto službu. Poté již klient provede požadovanou operaci přímo s danou službou.

Službu, která bude hledat optimální službu A pro klienta, nazveme službou I (Index Service). Předpokládá se, že služba A má po vytvoření určitý kontakt na službu I. Pomocí tohoto kontaktu se služba A zaregistruje u služby I. Pro urychlení prvotní komunikace může služba A společně s registrací poslat službě I i svá statická data. Následně přicházejí v úvahu dva scénáře:

Update na vyžádání - v tomto scénáři se při každém dotazu na optimální službu od klienta ptá služba I všech služeb A na jejich dynamická data. Poté služba I vybere optimální službu A a pošle na ní odkaz klientovi.

Tento scénář je vhodný, pokud by byla vysoká frekvence dotazů od klientů a pokud by služba I stahovala data od služeb po uplynutí předem nastaveného timeoutu. Je tím myšleno, že by nestahovala data při každém dotazu od klienta, ale pouze pokud by dotaz přišel po určité době od poslední aktualizace statických dat. Tím by se zabránilo možnému

zahlcení sítě, které by mohlo nastat při častém stahování velkého množství informací. Pokud by byla frekvence dotazů od klientů nízká, data by se stahovala při každém dotazu klienta. Vhodnost tohoto scénáře se také zvyšuje (snižuje) s rostoucí (klesající) frekvencí změn dynamických dat ve službách A a s množstvím služeb A, které by byly zaregistrovány u služby I.

Tento scénář má výhodu v tom, že síť není zatížena neustálým posíláním dynamických dat všem klientům a data se stahují pouze jednou službou I. Nevýhoda je v tom, že dynamická data, která obsahuje služba I, nemusejí být aktuální a tudíž i rozhodnutí služby I nemusí být pro klienta optimální.

Automatický update - Zde se narozdíl od předchozího případu posílají dynamická data automaticky od služby A ke službě I při každé změně těchto dat.

Uvedený scénář má výhodu oproti předchozímu scénáři v menší zátěži sítě, protože dynamická data se posílají pouze jednou při změně. To ovšem platí, pokud se takovéto změny nedějí příliš často, jinak se z toho stane nevýhoda. Další výhoda nastává při vysoké frekvenci dotazů od klientů, protože se minimalizuje zátěž při posílání updatů dynamických dat. Poslední výhoda spočívá v tom, že služba I má vždy aktuální verzi dat.

Tyto dva výše uvedené scénáře mají jednu společnou nevýhodu. Ta nastává, pokud by se ke službě I připojovalo několik verzí klientů a každý klient by měl jinak nastaveny priority pro rozhodování, která služba je pro něho výhodnější. Každý klient může upřednostňovat zcela odlišné vlastnosti. Pak by služba I neprováděla optimální rozhodnutí pro všechny klienty.

5.2.2 Rozhoduje klient

Zde provádí rozhodnuí o optimálním výběru služby klient. Klient se nejprve připojí ke službě I, která mu pošle nějaká data, na jejichž základě klient vybere nejlepší službu. V tomto případě jsou možné následující 2 scénáře:

Update na vyžádání klientem - klient po dotazu službě I obdrží seznam služeb A. Poté se k těmto službám připojí, stáhne si jejich dynamická data, vybere nejlepší službu a poté s ní provede požadovanou operaci.

Zde je výhoda v tom, že klient si vybere nejlepší službu podle aktuálních dat a svých preferencí. Nevýhoda nastává při častých dotazech velkého počtu klientů nebo při velkém počtu služeb. V tomto případě může být síť neúměrně zatížena dotazy klientů na dynamická data služeb. Pokud by se měl klient dotazovat služeb A příliš často, bylo by vhodné stanovit dobu, po kterou by klient používal svá stará data.

Tento scénář lze optimalizovat tak, že klient pošle při úvodním dotazu službě I seznam požadavků a služba I mu vrátí pouze ty služby A, které vyhovují požadavkům od klienta. Tím lze vyřadit služby, které by si klient pravděpodobně stejně nevybral. Takto lze zmenšit zatížení sítě a čas pro nalezení optimální služby pro klienta.

Automatický update - Tento scénář vychází ze stejnojmenného scénáře, kdy vybírala optimální službu služba I. Služba I přijímá pravidelné updaty při změně dynamických dat od služeb A. Při dotazu klienta je zpět poslán seznam služeb se statickými a dynamickými daty. Klient poté podle nich vybere nejlepší službu. I zde lze zavést optimalizaci, kdy služba I vrátí pouze seznam těch služeb, které budou vyhovovat prvotním požadavkům od klienta.

Výhoda tohoto scénáře spočívá v tom, že síť je ušetřena dotazů na dynamická data od všech klientů. Výběr optimální služby navíc proběhne podle všech preferencí daného klienta s aktuálními daty. Nevýhoda nastává, pokud by se dynamická data měnila příliš často a také, pokud by služba I měla informace o velkém množství služeb A. Pak by se mohlo stát, že by server, kde je umístěna služba I, nebyl schopen zpracovat ohromný příval updatů a dotazů od klientů a odeslat jim velké objemy dat.

V případě, kdy optimální službu vybírá klient, lze tento výběr provést přesně podle aktuálních preferencí klienta. Zde však může narůst na neúnosnou míru zatížení sítě během zjišťování dynamických dat od více klientů.

Shrnutí vlastností výše uvedených scénářů je zobrazeno v tabulce 5.1 na straně 39, kde zkratka *S update* znamená scénář, kdy rozhoduje služba a update dat je na vyžádání. Zkratka *S auto* představuje scénář, kdy rozhoduje služba a update dat se provádí automaticky. Obdobně jsou značeny scénáře, kdy rozhoduje klient (*K update* a *K auto*).

	+	-
S update	při vysoké frekvenci dotazů od klientů	nemusí rozhodnout nejlépe
	při vysoké frekvenci změn dyn. dat	při nízké frekvenci změn dyn. dat
	data se stahují pouze 1x	data nemusí být aktuální
S auto	nižší zátěž sítě při nízké frekvenci změn dynamických dat	vyšší zátěž sítě při vysoké frekvenci změn dynamických dat
	při vyšší frekvenci dotazů od klientů	nemusí rozhodnout nejlépe
	vždy aktuální verze dat	
K update	nejlepší rozhodnutí	při vysokém počtu klientů s častými dotazy
	lze provést prvotní výběr služeb	při vysokém počtu služeb
K auto	nižší zátěž sítě při nízké frekvenci změn dynamických dat	vyšší zátěž sítě při vysoké frekvenci změn dynamických dat
	lze provést prvotní výběr služeb	posílání velkého množství dat klientům
	nejlepší rozhodnutí	

Tabulka 5.1: Přehled vlastností scénářů

Kapitola 6

Implementace

V minulé kapitole byly popsány různé způsoby, jak přistupovat k interakci grid-mobilní klient. V této kapitole je popsána implementace jednoho z dříve uvedených scénářů.

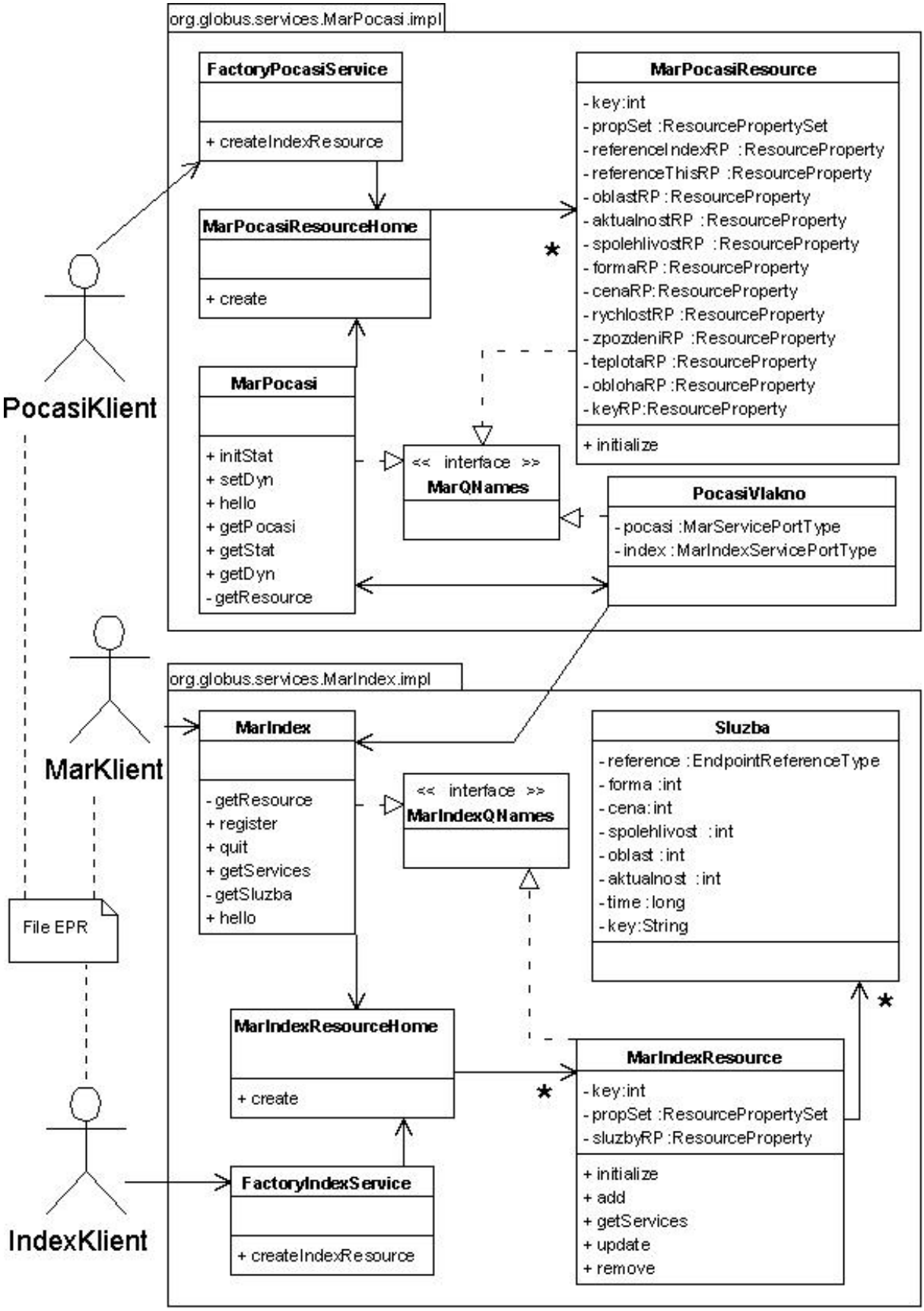
6.1 Implementační model

Pro praktickou ukázku spolupráce mobilního klienta s gridem jsem se rozhodl naprogramovat jednoduchou službu, která vrací text aktuálního počasí v několika městech. Tato služba slouží pro demonstrační účely a vrací počasí ve formáu HTML nebo Text. Spolu s dalšími parametry si klient takto volí, jaký chce dostat výstup, protože jak již bylo uvedeno v minulé kapitole, každý klient má různé technické prostředky. Jako scénář jsem si zvolil způsob, kdy optimální službu vybírá mobilní klient (scénář *Update na vyžádání* v kapitole 5.2.2 na straně 37). Tento scénář byl zvolen, protože klient potřebuje najít nejlepší službu a předpokládá se, že se dynamická data budou měnit poměrně často. Na obrázku 6.1 na straně 41 je zobrazen implementační model.

V následujících kapitolách bude služba, která představuje index službu pro správu služeb vracejících počasí nazývána *IndexService* a služba vracející počasí *PocasiService*. Ve zdrojových souborech jsou u každé služby pojmenovány jinými názvy ty části, které spolu přímo nesouvisejí. Důvodem k tomu je přehledné oddělení na sobě nezavislých částí.

Tento model obsahuje 3 části:

- PocasiService (PS) - implementace služby, která vrací klientovi počasí v jednotlivých městech ve formátu text nebo HTML



Obrázek 6.1: Implementační model

- IndexService (IS) - implementace index služby, která se stará o registraci jednotlivých PocasiService a klientovi posílá jejich seznam

- kódy tříd klientů, které spouštějí výše uvedené služby

6.1.1 Proces činnosti modelu

Celý proces spuštění služeb a získání počasí v nějakém městě vypadá takto:

1 - IndexClient vytvoří nový zdroj pro službu *IndexService* (IS) a do souboru *fileEPR* (jméno souboru závisí na parametru, který je předán klientovi při spuštění) uloží *Endpoint Referenci* (EPR) na tuto dvojici služba-zdroj (WS-Reference).

2 - PocasiClient načte ze souboru fileEPR EPR na IS, vytvoří nový zdroj pro *PocasiService* (PS) a pošle PS staticka data + EPR na IS. Poté spustí vlákno, které ovládá aktualizace dynamických dat služby. Tím se simuluje změna dynamických dat externí entitou a při každé aktualizaci dat se také náhodně změní počasí a teplota ve všech městech.

3 - PS uloží statická data od klienta a spustí vlákno *PocasiVlakno*. Tomuto vláknu předá mimo jiné EPR na IS a sebe sama.

4 - PocasiVlakno nejprve zaregistruje PS u IS (každá PS služba má jedinečný klíč skládající se z ID zdroje + IP adresy služby) a‘poté oběma službám posílá v předem nastaveném časovém intervalu zprávy (hello), kterými zjišťuje, zda stále existuje spojení mezi IS a PS.

5 - IS si po obdržení registrace PS uloží službu do svého seznamu služeb. Po každém příjmu hello zprávy od *PocasiVlakno* si u příslušné služby aktualizuje časový údaj o příjmu poslední zprávy od dané služby.

6 - MarClient se po načtení EPR na IS ze souboru fileEPR připojí k IS a požádá jí o poskytnutí seznamu služeb, které vyhovují intervalům statických dat předanými IS při žádosti.

7 - IS vybere ze seznamu služeb takové služby, které vyhovují zadaným intervalům a jejichž poslední update přišel do určité doby, a nakonec pošle seznam zpět MarClientovi.

8 - MarClient po obdržení seznamu služeb kontaktuje každou službu ze seznamu, stáhne si její aktuální data a vypočítá váhu služby, podle které následně určí nejlepší službu.

9 - MarClient se připojí k nejlepší službě a zjistí,jaké počasí je v daném městě.

6.1.2 Zpracování chyb

V implementaci je několik mechanismů, které zpracují chybné stavy, do kterých se služby mohou dostat. Jelikož *PocasiVlakno* běží samostatně, tak je schopno detekovat nedostupnost IS nebo PS. Pokud je nedostupná PS, vlákno odregistruje PS u IS. Pokud je naopak nedostupná IS, vlákno zruší příslušný zdroj u PS. Ještě může nastat případ, kdy se z nějakého důvodu spolu s PS ukončí i běh vlákna PocasiVlakno, a proto si IS ke každé službě v seznamu ukládá čas posledního kontaktu od dané služby (registrace služby nebo příchod poslední hello zprávy). Při žádosti od *MarClienta* pošle IS pouze seznam takových služeb, jejichž poslední kontakt přišel do nějakého časového intervalu. Zbylé služby IS ze svého interního seznam vyřadí.

V následující části je podrobněji popsána služba *PocasiService*.

6.2 Služba PocasiService

V této sekci bude podrobněji popsán obsah souborů, které jsou nutné pro správnou funkci každé gridové služby. Jako příklad jsou použity zdrojové soubory pro službu *PocasiService*. U souborů budou ukázány pouze nejdůležitější části.

6.2.1 Soubor MarService.wsdl

Podle tohoto souboru jsou při překladu vygenerovány zdrojové soubory spojek v Javě. Zde se specifikuje, jaké *public* metody bude mít služba, jejich parametry a také vlastnosti zdrojů.

Soubor se zkládá ze 4 částí (úvodní definice, types, messages a portType) a začíná root elementem *definitions*. Podle hodnoty parametru *name* se pojmenují části některých spojek v adresáři *SERVICE.service.XX* (viz překlad služby v příloze A na straně 63. Jako další parametry jsou zde jmenné prostory *namespaces*, z nichž nejdůležitější je

targetNamespace:

```
<definitions name="MarousekWeatherService"
 targetNamespace="http://www.globus.org/namespaces/MarousekPocasiService"
 xmlns:wsrlw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-Resource
 Lifetime-1.2-draft-01.wsdl"
      ........
```

Hodnota u *targetNamespace* se dále využívá při specifikaci místa pro uložení vygenerovaných spojek. Ve službě budeme využívat metodu *destroy()* ze specifikace *WS-Resource Lifetime*, a proto je nutné nadefinovat příslušný soubor, kde je tato metoda popsána.

```
<wsdl:import
namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-Resource
Lifetime-1.2-draft-01.wsdl"
location="../wsrf/lifetime/WS-ResourceLifetime.wsdl"/>
```

Dále obsahuje soubor *MarService.wsdl* specifikaci typů *types*, které se používají jako parametry při volání všech služeb. Zde je uveden příklad specifikace vstupního parametru pro službu *getDyn()* a výstupního pro metodu *getID()*:

```
<xsd:element name="getDynInput">
          <xsd:complexType/>
</xsd:element>
<xsd:element name="getIDOutput" type="xsd:string"/>
```

Typ, který vrací metoda, je *long* a vstupní parametr metody je *void*. Ve WSDL je nutné všechny parametry *void* specifikovat jako *complexType*. To platí i v případě, kdy má služba více něž jeden primitivní parametr. V případě, kdy chceme definovat například 2 parametry typu int, je třeba toto provést následovně:

```
<xsd:element name="dynType"> <xsd:complexType> <xsd:sequence>
     <xsd:element name="rychlost" type="xsd:int"/>
     <xsd:element name="zpozdeni" type="xsd:int"/>
</xsd:sequence> </xsd:complexType> </xsd:element>
```

Při překladu bude poté automaticky vygenerována třída *DynType* se dvěma atributy *rychlost* a *zpozdeni* a budou také vygenerovány metody *setRychlost* a‘*getRychlost*. Instance této třídy se následně použije pro volání metody *getDyn(DynType)* (viz dále).

Jako poslední část elementu *types* je definice resource propeties.

```
<xsd:element name="Oblast" type="xsd:int"/>
<xsd:element name="Aktualnost" type="xsd:int"/>

<xsd:element name="MarResourceProperties"> <xsd:complexType> <xsd:sequence>
     <xsd:element ref="tns:Oblast" minOccurs="1" maxOccurs="1"/>
     <xsd:element ref="tns:Aktualnost" minOccurs="1" maxOccurs="1"/>
        ..............
```

Jako statická data obsahuje služba:

- Forma: 0=Text, 1=HTML
- Oblast: 1-9, kde 1 je nejlepší
- Cena: 1-9, kde 1 je nejlepší
- Spolehlivost: 1-9, kde 1 je nejlepší
- Aktualnost: 1-9, kde 1 je nejlepší

Jako dynamická data obsahuje služba:

- Rychlost: 1-9, kde 1 je nejlepší
- Zpozdeni: 1-9, kde 1 je nejlepší

Po elementu *types* následují specifikace zpráv:

```
<message name="GetDynInputMessage">
      <part name="parameters" element="tns:getDynInput"/>
</message>
<message name="GetDynOutputMessage">
     <part name="parameters" element="tns:dynType"/>
</message>
```

Tyto zprávy se použijí v posledním elementu *potType*:

```
<portType name="MarServicePortType"
     wsrp:ResourceProperties="tns:MarResourceProperties"
```

```
        wsdlpp:extends="wsrlw:ImmediateResourceTermination
             ........
        <operation name="getDyn">
             <input message="tns:GetDynInputMessage"/>
             <output message="tns:GetDynOutputMessage"/>
        </operation>
             ........
    </portType>
```

Hodnota parametru *name* musí odpovídat jménu souboru+PortType. V příkladu se využívá *WSDL preprocesoru*, který sám dosadí metody definované v souboru z prostoru jmen *wsrlw*. Jméno metody, která se bude nakonec používat u služby, je specifikováno u atributu *name* elementu *operation*.

6.2.2 Soubory *.java

Tyto soubory obsahují zdrojový kód pro službu. Jako parametry metod se musí použít spojky, které byly vygenerovány z příslušného .wsdl souboru.

Služba PocasiService obsahuje tyto .java soubory:

FactoryPocasiService.java - tento soubor představuje implementaci *factory*, která obsahuje pouze jednu metodu *createResource()*. Tato metoda vytvoří nový zdroj voláním metody *create()* třídy *MarPocasiResourceHome* a vrací EPR na dvojici služba+nový zdroj.

MarPocasiResourceHome.java - implementace ResourceHome. Obsahuje metodu *create()*, která vytvoří a inicializuje novou instanci třídy *MarPocasiResource*.

MarPocasiResource.java - tato třída představuje jednotlivé zdroje. Kromě metody *initialize()* obsahuje get/set metody pro přístup k jednotlivým atributům zdroje.

MaPocasi.java - implementace služby. Obsahuje metodu *getResource()*, která vrací odkaz na objekt třídy *MarPocasiResource*. Dále jsou zde veřejné metody, které implementují očekávanou funkci služby.

MarQNames.java - toto rozhraní obsahuje proměnné třídy, které se využívají ve zbylých souborech.

MyFormat.java - pomocná třída sloužící k nastavení formátu pro logování.

PocasiVlakno.java - zde je implementováno vlákno spouštěné službou *PocasiService*, které v nastavených intervalech posílá hello zprávy *PocasiService* a *IndexService*.

Dále budou uvedeny nejzajímavější úseky z výše uvedených souborů.

FactoryPocasiSevice.java - Zde se použije třída *ResourceContext* k nalezení *resource home*, který přísluší k dané službě a pomocí něhož se vytvoří nový zdroj. Ten je identifikován klíčem (proměnná *key*), který se následně použije k vytvoření EPR na službu *PocasiService*.

```
ctx = ResourceContext.getResourceContext();
home = (MarPocasiResourceHome) ctx.getResourceHome();
key = home.create();
     ........
URL baseURL = ServiceHost.getBaseURL();
String instanceService = (String) MessageContext.getCurrentContext()
   .getService().getOption("instance");
String instanceURI = baseURL.toString() + instanceService;
epr = AddressingUtils.createEndpointReference(instanceURI, key);
```

EPR je nakonec vrácen klientovi, který se s ním připojí k dané službě.

MarPocasiResourceHome.java - Zde je v metodě *create()* vytvořena nová instance zdroje, která je následně přidána do seznamu zdrojů, a zpět je vrácen klíč nově vytvořeného zdroje. Proměnná *keyTypeName* je protected proměnná obsahující typ klíče zdroje.

```
MarPocasiResource res = (MarPocasiResource) createNewInstance();
res.initialize();
ResourceKey key = new SimpleResourceKey(keyTypeName, res.getID());
add(key, res);
return key;
```

MarResource.java - Tato třída představuje jednotlivé zdroje. Zde je uložen seznam resource properties (RP, proměnná *propSet*), který obsahuje jednotlivé RP. Ty jsou identifikovány pomocí statických proměnných z rozhraní *MarQNames*.

```
private ResourcePropertySet propSet;
 .....
private ResourceProperty rychlostRP, zpozdeniRP;
 .....
this.propSet = new SimpleResourcePropertySet(RESOURCE_PROPERTIES);
 .....
rychlostRP = new SimpleResourceProperty(RP_RYCHLOST);
rychlostRP.add(new Integer(0));
 .....
this.propSet.add(rychlostRP);
```

Dále jsou zde metody get/set pro práci se všemi RP.

MarPocasi.java - Tato třída představuje implementaci služby dle portTypu specifikovaného v souboru *MarService.wsdl*. Metoda *getResource()* vrací odkaz na instanci třídy *MarResource* a pomocí metod get/set ze třídy *MarResource* se pracuje s atributy zdroje. Tato třída při inicializaci statických dat spustí vlákno *PocasiVlakno*, které je popsáno níže. Služba umí vracet počasí ve dvou formátech: text nebo HTML.

PocasiVlakno.java - Toto vlákno má EPR na instance tříd *MarPocasi* (PS) a *MarIndex* (IS). Po spuštění zaregistruje službu PS u IS a poté po nastaveném časovém intervalu posílá zprávy (volá metody) službám IP a PS. Pokud se nemůže spojit s PS, volá vlákno metodu *quit()* IS. Pokud se nemůže spojit s IS, volá metodu *destroy()* PS, která zruší zdroj PS.

6.2.3 deploy-server.wsdd

Tento soubor se nazývá *deployment descriptor* a slouží ke správnému zavedení služby do gridového kontejneru.

```
<service name="MarousekPocasiService"  ...........
   <parameter name="className" value="org.globus.services.MarPocasi.impl.MarPocasi"/>
   <wsdlFile>share/schema/PocasiService/MarService_service.wsdl</wsdlFile>
```

Atribut *name* u elementu *service* určuje výslednou adresu, pod kterou bude služba v kontejneru přístupná. Tato hodnota se přidá k adrese kontejneru. Atribut *name* u elementu *parameter* určuje, která třída v Javě představuje tuto službu, a nakonec element

wsdlFile určuje umístění .wsdl souboru, který popisuje *portType* dané služby. U služby, která funguje jako *factory* je navíc následující řádek, který určuje, pro jakou službu je daná factory určena.

```
<parameter name="instance" value="MarousekPocasiService"/>
```

6.2.4 deploy-jndi-config.xml

Obsah tohoto souboru sděluje kontejneru, které třídy představují *resourceHome* a *resource* pro danou službu a factory. Je zde také uveden typ klíče, kterým je každý zdroj identifikován.

```
<service name="MarousekPocasiService">
  <resource name="home" type="org.globus.services.MarPocasi.impl.MarPocasiResourceHome">
  <resourceParams>
    <parameter> <name>resourceClass</name>
      <value>org.globus.services.MarPocasi.impl.MarPocasiResource</value>
    </parameter>
    <parameter> <name>resourceKeyType</name>
      <value>java.lang.Integer</value>
    </parameter>
       ......
```

Jméno služby musí být stejné, jako je uvedeno v souboru *deploy-server.wsdd*. Hodnota u parametru *resourceKeyName* slouží k identifikaci zdroje.

6.3 Služba IndexService

Tato služba vychází ze stejné architektury jako služba *PocasiService*. Z tohoto důvodu jsou prakticky stejné soubory pro deployment služby do kontejneru (liší se pouze jména služeb a jména tříd, které danou službu, resource a resourceHome implementují). Ve .wsdl souboru se specifikací *portTypu* pro IndexService jsou definovány rozsáhlejší struktury pro posílání kompletních informací o službě a samozřejmě jiné metody. Jako příklad je uvedena struktura *serviceType*, která slouží pro příjem registračních dat od služby PS (respektive vlákna *PocasiVlakno*, které službu PS zastupuje).

```
<xsd:element name="serviceType">
```

```
 <xsd:complexType>
  <xsd:sequence>
      <xsd:element name="key" type="xsd:string"/>
      <xsd:element name="reference" ref="wsa:EndpointReference"/>
      <xsd:element name="forma" type="xsd:int"/>
      <xsd:element name="oblast" type="xsd:int"/>
      <xsd:element name="aktualnost" type="xsd:int"/>
      <xsd:element name="cena" type="xsd:int"/>
      <xsd:element name="spolehlivost" type="xsd:int"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
```

Implementace factory a resource home jsou také prakticky stejné. Třída *MarIndexResource* (představuje zdroj pro službu IndexService) se liší od třídy *MarResource* v tom, že obsahuje *HashMap*, do které se ukládají zaregistrované služby PocasiService. Jako klíč pro uložení služby do mapy se bere identifikátor, skládající se z čísla zdroje a adresy kontejneru, ve kterém služba běží. Tato třída dále obsahuje metody pro přidání, nalezení a odebrání záznamu z hashmapy. Třída *MarIndex* obsahuje metody, které slouží pro registraci (deregistraci) služby PS a příjem hello zpráv od *PocasiVlakno*. Je zde také metoda, která vrací klientovi seznam služeb, které vyhovují prvotnímu kritériu pro výběr.

V balíků tříd pro službu IndexService se nachází třída *Sluzba*, která reprezentuje jednotlivé služby PocasiService, jež jsou uloženy v hashmapě.

Z důvodu hodně podobného kódu se službou PocasiService zde nebude uveden podrobný výpis jednotlivých částí souborů, jako tomu bylo v případě PocasiService.

Více informací o tvorbě služeb v GT4 lze nalézt zde [11], [12] nebo na stránkách projektu Globus [13].

6.4 Klienti

V modelu jsou implementováni 3 klienti:

IndexClient - tento klient je určen pro spuštění IndexService

PocasiClient - klient pro PocasiService

MarClient - tento klient kontaktuje IndexService a poté vybere z vráceného seznamu nejlepší službu

V následující části jsou popsány podstatné pasáže z výše uvedených klientů.

6.4.1 IndexClient.java

Tento klient po zavolání factory pro *IndexService* obdrží endpoint referenci na IndexService, kterou uloží do souboru, jehož jméno je jako parametr klienta. Nalezení factory a získání EPR na IndexService se provede takto:

```
factoryEPR = new EndpointReferenceType();
factoryEPR.setAddress(new Address(factoryURI));
factory = factoryLocator.getFactoryIndexPortTypePort(factoryEPR);

CreateIndexResourceResponse createResponse = factory
    .createIndexResource(new CreateIndexResource());
instanceEPR = createResponse.getEndpointReference();
```

Zde je převod EPR na String a jeho uložení do souboru:

```
String endpointString = ObjectSerializer.toString(instanceEPR,RESOURCE_REFERENCE);
FileWriter fileWriter = new FileWriter(fileName);
BufferedWriter bfWriter = new BufferedWriter(fileWriter);
bfWriter.write(endpointString);
bfWriter.close();
```

6.4.2 MarClient.java

Tento klient načte ze souboru EPR na IndexService, poté se k této službě připojí, obdrží seznam služeb, stáhne si jejich dynamická data a nakonec od nejlepší služby obdrží počasí ve vybraném městě. Načtení a získání reference na IndexService se provede takto:

```
FileInputStream fis = new FileInputStream(fileName);
indexEPR = (EndpointReferenceType) ObjectDeserializer.deserialize(
    new InputSource(fis), EndpointReferenceType.class);
fis.close();
```

```
MarIndexServicePortType index = instanceIndexLocator
   .getMarIndexServicePortTypePort(indexEPR);
```

Proces od získání seznamu služeb z IndexService do obdržení počasí od PocasiService je ve zkratce popsán zde (definice typu *ServiceType* je uvedena v kapitole 6.3 na straně 49):

```
PoleSluzba pom = index.getServices(ser);
ServiceType[] pole = pom.getItem();
pocasi = instancePocasiLocator.getMarServicePortTypePort(pole[i].getReference());
DynType dyn = pocasi.getDyn(new GetDynInput());
String poc = pocasi.getPocasi(MESTO_CHOMUTOV);
```

6.5 Testování a výsledky

Testování bylo provedeno na dvou počítačích s IP adresami 192.168.0.11 (stroj A) a 192.168.0.5 (stroj B). Kontejner gridových služeb byl spustěn na stroji A. V následujícím výpisu kódu je vidět postup při vytváření zdroje pro IndexService (Factory-ResourceHome-Resource):

```
21.08.2005  18:35:27.026   FactoryIndexService   createResource
FINER:  ENTRY
21.08.2005  18:35:27.028   MarIndexResourceHome   create
FINER:  ENTRY
21.08.2005  18:35:27.028   MarIndexResource   initialize
FINER:  ENTRY
21.08.2005  18:35:27.029   MarIndexResource   initialize
INFO:  HashCode noveho zdroje: 17621911
21.08.2005  18:35:27.054   MarIndexResource   initialize
FINER:  RETURN
21.08.2005  18:35:27.054   MarIndexResourceHome   create
FINER:  RETURN
21.08.2005  18:35:27.056   FactoryIndexService   createResource
FINER:  RETURN
```

Na strojích A a B byly spuštěny tyto služby (u Formy 1=HTML, 0=Text):

Po vytvoření byly služby zaregistrovány u služby IndexService (IS),

Stroj	Hash	Forma	Cena	Oblast	Spolehlivost	Aktuálnost	Zkratka
A	2826327	1	5	5	5	5	S1
A	26507463	1	1	8	4	5	S2
A	2559940	0	1	8	4	5	S3
A	3695078	0	4	6	4	5	S4
B	12476680	0	4	6	4	5	S5
B	23803180	1	4	6	4	5	S6

Tabulka 6.1: Přehled služeb použitých v testu

```
21.08.2005  18:37:29.817   MarIndexResource   add
INFO:  17621911   Do seznamu byla pridana sluzba s ID:2826327:
http://192.168.0.11:8080/wsrf/services/MarousekPocasiService

21.08.2005  18:39:24.058   MarIndexResource   add
INFO:  17621911   Do seznamu byla pridana sluzba s ID:12476680:
http://192.168.0.5:8080/wsrf/services/MarousekPocasiService
```

v pravidelných 10s intervalech přicházely hello zprávy a u služeb byl prováděn update posledního kontaktu:

```
21.08.2005  18:37:39.913   MarIndex   hello
FINER:  17621911   Prislo HELLO
21.08.2005  18:37:39.914   MarIndexResource   update
FINER:  17621911   Probehl UPDATE sluzby:2826327:
http://192.168.0.11:8080/wsrf/services/MarousekPocasiService

21.08.2005  18:39:34.111   MarIndex   hello
FINER:  17621911   Prislo HELLO
21.08.2005  18:39:34.112   MarIndexResource   update
FINER:  17621911   Probehl UPDATE sluzby:12476680:
http://192.168.0.5:8080/wsrf/services/MarousekPocasiService
```

Poté se k IS několikrát připojil MarClient s těmito požadavky:

Jako výsledek připojení A obdržel klient tyto údaje (z důvodů přehlednosti výstupů byly vymazány poslední části klíčů služeb):

Připojení	Forma	Cena	Oblast	Spolehlivost	Aktuálnost
A,C	1	3-9	2-9	3-9	2-7
B	0	3-9	2-9	3-9	2-7

Tabulka 6.2: Přehled připojení klientem

```
Zpozdeni: 1  Rychlost: 1 Vaha: 2 2826327:http://192.168.0.11:
Zpozdeni: 5  Rychlost: 6 Vaha: 11 23803180:http://192.168.0.5:
Vyhrala sluzba na pozici: 1 s vahou: 2
Zjistuji pocasi ve meste Chomutov od sluzby s nejlepsi vahou ..... hotovo
Vysledek: <HTML><HEAD></HEAD><BODY>Soucasne pocasi ve meste Chomutov je:
 28 stupnu a Zatazeno</BODY></HTML>
           .............  15s interval  ...........
Zpozdeni: 2  Rychlost: 4 Vaha: 6 2826327:http://192.168.0.11:
Zpozdeni: 1  Rychlost: 3 Vaha: 4 23803180:http://192.168.0.5:
Vyhrala sluzba na pozici: 2 s vahou: 4
Zjistuji pocasi ve meste Chomutov od sluzby s nejlepsi vahou ..... hotovo
Vysledek: <HTML><HEAD></HEAD><BODY>Soucasne pocasi ve meste Chomutov je:
 16 stupnu a Polojasno</BODY></HTML>
```

Požadavek na formu výstupu byl formát *HTML*. Z tohoto důvodu nebyly od IndexService nabídnuty klientovi služby S3, S4 a S5. Služba S2 nesplňuje požadavek na *Cenu*, a proto byla také vynechána. Klient si z obdržených služeb zjistil jejich váhy a poté si vybral nejlepší službu (s nejnižší váhou). Po intervalu 15s připojení opakoval a dostal stejné služby s jinými dynamickými daty. Podobně jako v případě A, i v případě B (formát Text) byly vynechány služby, které nesplňují daný formát (S1, S2 a S6), a služba, kde nevyhovuje Cena (S3):

```
Zpozdeni: 3  Rychlost: 2 Vaha: 5 3695078:http://192.168.0.11:
Zpozdeni: 1  Rychlost: 5 Vaha: 6 12476680:http://192.168.0.5:
Vyhrala sluzba na pozici: 1 s vahou: 5
Zjistuji pocasi ve meste Chomutov od sluzby s nejlepsi vahou ..... hotovo
Vysledek: Soucasne pocasi ve meste Chomutov je: 10 stupnu a Tornado
           .............  15s interval  ...........
Zpozdeni: 8  Rychlost: 7 Vaha: 15 3695078:http://192.168.0.11:
Zpozdeni: 3  Rychlost: 5 Vaha: 8 12476680:http://192.168.0.5:
Vyhrala sluzba na pozici: 2 s vahou: 8
```

```
Zjistuji pocasi ve meste Chomutov od sluzby s nejlepsi vahou ..... hotovo
Vysledek: Soucasne pocasi ve meste Chomutov je: 27 stupnu a Tornado
```

Poté byla zrušena služba S1. Z výpisu je vidět, že *PocasiVlakno* od tohoto zdroje se nemohlo spojit s *PocasiService* (PS), a proto odregistrovalo tuto službu u IS.

```
21.8.2005 18:44:43 org.globus.services.MarPocasi.impl.PocasiVlakno run
SEVERE: 2826327   Nelze se spojit s PocasiService

21.8.2005 18:44:43 org.globus.services.MarIndex.impl.MarIndexResource remove
INFO: 17621911   Ze seznamu byla odebrana sluzba s ID:2826327:
http://192.168.0.11:8080/wsrf/services/MarousekPocasiService
```

Když se poté připojil klient k IS (připojení C), obdržel tento výsledek (stejný jako v připojení A, ovšem bez zrušené služby S1):

```
Zpozdeni: 3  Rychlost: 8 Vaha: 11 23803180:http://192.168.0.5:
Vyhrala sluzba na pozici: 1 s vahou: 11
Zjistuji pocasi ve meste Chomutov od sluzby s nejlepsi vahou ..... hotovo
Vysledek: <HTML><HEAD></HEAD><BODY>Soucasne pocasi ve meste Chomutov je:
 33 stupnu a Tezky dest</BODY></HTML>
```

Z výše uvedených výpisů jsou patrné základní komunikace mezi komponenty a funkčnost klienta. V reálných situacích může v systému nastat několik chybových stavů, které jsou v implementaci také ošetřeny. Zde byla na jednoduchém příkladu ukázána pouze základní funkčnost.

Kapitola 7

Závěr

V úvodních kapitolách této práce je uveden teoretický základ ke gridové problematice. Jsou zde popsány nejdůležitější projekty z oblasti gridů a lze si tak představit, k čemu všemu se dají gridy využít. Poté je zde popsán Globus Toolkit a tvorba služeb. V poslední části práce je po teoretickém prostudování několika možností, jak spolu může spolupracovat grid a mobilní klient, jeden takovýto případ realizován.

Realizace byla provedena pro demonstrační účely na příkladu jednoduché služby (Pocasi), která slouží pro zjištění aktuálního počasí v několika městech . Těchto služeb může být více, a proto bylo nutné vytvořit další službu (Index), která se stará o jejich správu. Klient poté komunikuje pouze s Index službou. Jelikož byla implementace vytvořena pouze pro demonstrační účely, v této podobě nemá praktické využití. S určitými dílčimi úpravami lze ale tento model úspěšně použít v reálných situacích.

Aby se mohla služba Pocasi zaregistrovat u Index služby a klient se poté mohl k Index službě připojit, je nutné informace o Index službě nějakým způsobem zveřejnit. V této práci jsem použil způsob uložení do souboru. Klient a Pocasi služba si poté tyto informace ze souboru načtou a spojí se s Index službou. V tomto jednoduchém příkladu je uvedený způsob postačující, ale v rozsáhlejších projektech by bylo nutné nějakým způsobm vyřešit distribuci informací nezbytných k připojení se k Index službě, a to například přes www rozhraní. Toto vidím jako drobný nedostatek mého řešení, ale distribuce výše uvedených informací nebyla v zadání této práce.

S gridy jsem se poprvé setkal až v této diplomové práci, trvalo mi poměrně dlouhou dobu, než jsem částečně pronikl do jejich problematiky, a to výhradně díky zahraniční literatuře, neboť v českém jazyce neexistuje téměř žádný zdroj. Jsem si vědom toho, že ne

všechny konstrukce, které jsem v práci použil, jsou optimální. Nicméně jsem dospěl k závěru, že se jedná o rozsáhlou a zajímavou oblast, která si zaslouží mimořádnou pozornost zejména pro její praktické uplatnění.

Seznam použitých zkratek

EPR Endpoint Reference - reference na určitou službu + zdroj. Definována specifikací WSRF.

GS - Grid Service - gridová služba. Založena na principu webových služeb.

GSI - Grid Security Infrastructure - implementace zabezpečení přenosů pomocí kryptografických metod.

GTx - Globus Toolkit - nejzdařilejší implementace OGSA, OGSI (verze 3) a WSRF (verze 4). Základ pro model v této práci.

OGSA - Open Grid Service Architecture - vrstevnatá architektura specifikující funkčnost gridových služeb.

OGSI - Open Grid Services Infrastructure - standardizace infrastruktury gridového SW za účelem dosáhnutí co největší interoperability mezi OGSA komponenty.

HTML - HyperText Markup Language - značkovací jazyk používající se v síti Internet pro popis vzhledu stránek.

PORTTYPE - rozhraní - definice metod, pomocí kterých lze přistupovat k dané službě.

SDE - Service Data Element - objekt, který obsahuje stavové informace o gridové službě. Používá se v GT3.

SOAP - Simple Object Access Protocol - protokol, používaný v komunikaci mezi webovými službami.

WS - Web Service - webová služba. Základ pro gridové služby.

WSDL - Web Services Description Language - jazyk používaný pro popis webových a gridových služeb

WS-RESOURCE - WebService-Resource - dvojice služba+zdroj. Jednoznačné označení služby a zdroje, který daná služba využívá

WSRF - Web Services Resource Framework - společný standard pro webové a gridové služby

XML - Extensible Markup Language - jazyk pro popis dat

Literatura

[1] *Ian Foster, Carl Kesselman:* **The Grid:Blueprint for a New Computing Infrastructure**
Morgan Kaufmann Publishers; 1st edition (November 1, 1998), ISBN 1558604758

[2] *Ian Foster, Carl Kesselman:* **The Grid 2:Blueprint for a New Computing Infrastructure**
Morgan Kaufmann Publishers; 2nd edition, 2004, ISBN 1558609334

[3] *Ian Foster, Carl Kesselman, Steven Tuecke:* **The Anatomy of the Grid:Enabling Scalable Virtual Organizations**
International J. Supercomputer Applications, www.globus.org, 2001

[4] *Kleinrock L.:* **UCLA Press Release**
July 3, 1969

[5] *Joshy Joseph, Craig Felleinstein:* **Grid Computing**
IBM Press: On Demand Series, Prentice Hall, 2004, ISBN 0131456601

[6] *Ian Foster, Carl Kesselman, Steven Tuecke:* **The Physiology of The Grid: An Open Grid Services Architecture for Distributed Systems Integration**
http://www.globus.org/research/papers/ogsa.pdf

[7] *Ladislav Pešička:* **Interaction model of grid services in mobile grid environment**
www.kiv.zcu.cz/research/groups/dss/download/presentation-2004-10-22-e.ppt

[8] *Thomas Sandholm, Jarek Gawor:* **Gloobus Toolkit 3 Core - A Grid Service Container Framework**
http://www-unix.globus.org/toolkit/3.0/ogsa/docs/gt3_core.pdf

[9] *William E. Allcock, Ian Foster, Ravi Madduri:* **Reliable Data Transport: A Critical Service for the Grid**
http://www.doc.ic.ac.uk/%7Esjn5/GGF/GGF11/BGBS-Allcock.pdf

[10] *William E. Allcock, Ravi Madduri:* **Lessons learned producing an OGSI compliant Reliable File Transfer Service**
http://www-unix.mcs.anl.gov/%7Ekeahey/DBGS/DBGS_files/
dbgs_papers/allcock.pdf

[11] *Borja Sotomayor:* **The Globus Toolkit 4 Programmer's Tutorial**
http://gdp.globus.org/gt4-tutorial/

[12] *Terry Harmer, Julie McCabe:* **GT3.2 to GT4 Migration: A First HOWTO**
http://www.qub.ac.uk/escience/howtos/GT3 to GT4 Version 0.3.htm

[13] **The Globus Toolkit**
`http://www.globus.org`

[14] **The Global Grid Forum**
`http://www.ggf.org/`

[15] **Legion, A Worldwide Virtual Computer**
`http://legion.virginia.edu`

[16] **Condor, High Throughput Computing**
`http://www.cs.wisc.edu/condor`

[17] **Nimrod: Tools for Distributed Parametric Modelling**
`http://www.csse.monash.edu.au/ davida/nimrod/`

[18] **UNICORE (Uniform Interface to Computing Resources)**
`http://sourceforge.net/projects/unicore`

[19] **NSF Middleware Initiative**
http://www.nsf-middleware.org

[20] **DOE Science Grid**
http://www.doesciencegrid.org

[21] **EUROGRID - Application Testbed for European GRID computing**
http://www.eurogrid.org

[22] **The DataGrid Project**
http://eu-datagrid.web.cern.ch/eu-datagrid

[23] **EGEE: Enabling Grids for E-science in Europe**
http://egee-intranet.web.cern.ch/egee-intranet/gateway.html
http://egee.cesnet.cz

[24] **TeraGrid**
http://www.teragrid.org/

[25] **NASA Information Power Grid**
http://www.ipg.nasa.gov

[26] **SETI@home**
http://setiathome.berkeley.edu

[27] **United Devices Cancer Research Project**
http://www.grid.org/projects/cancer

Příloha A

Soubory služby a překlad

Adresářová struktura

Zde je popsáno umístění jednotlivých souborů nutných pro úspěšné přeložení služby. Jako kořenový adresář se bere adresář, kde je umístěn soubor *globus-build-service.sh*. Ve zdrojových souborech jsou u každé služby pojmenovány jinými názvy ty části, které spolu přímo nesouvisejí. Důvodem k tomu je přehledné oddělení na sobě nezavislých částí.

build.xml - soubor pro překlad služby pomocí Antu

globus-build-service.sh - build skript

namespace2package.mappings - mapování použitých jmenných prostorů na umístění vygenerovaných spojek:

```
http\://www.globus.org/namespaces/MarousekIndexService=
   org.globus.stubs.MarIndexService
http\://www.globus.org/namespaces/MarousekIndexService/bindings=
   org.globus.stubs.MarIndexService.bindings
http\://www.globus.org/namespaces/MarousekIndexService/service=
   org.globus.stubs.MarIndexService.service
```

build.mappings - tento soubor zjednodušuje spuštění build skriptu. Zde jsou uloženy parametry a skript si po spuštění tyto parametry sám načte. Jako první je uvedena zkratka pro spuštění skriptu, poté je zde umístění implementačních souborů (.wsdd,

impl/*.java, atd.) a nakonec umístění .wsdl souborů pro službu a její factory. Příklad souboru *build.mappings* pro spuštění build skriptu s parametrem *index*:

```
index,org/globus/services/MarIndex,schema/IndexService/MarIndexService.wsdl,
   schema/FactoryIndexService/FactoryIndex.wsdl
```

org/globus/services/MarIndex/deploy-server.wsdd - soubor pro umístění služby IndexService do kontejneru

org/globus/services/MarIndex/deploy-jndi-config.xml - soubor pro umístění služby IndexService do kontejneru

org/globus/services/MarIndex/impl/*.java - Java třídy pro službu IndexService

org/globus/services/MarPocasi/... - stejná adresářová struktura, tentokrát pro službu PocasiService

org/globus/clients/MarIndex/ - soubory pro spuštění klienta pro IndexSevice

org/globus/clients/MarService/ - soubory pro spuštění klienta pro PocasiService

org/globus/clients/MarClient/ - soubory pro spuštění klienta, který zjistí počasí

Překlad služby

Služba se překládá pomocí skriptu *globus-build-service.sh* (například příkaz "globus-build-service.sh index" pro *IndexService*). Při překladu je vytvořen adresář *build* (do stejného adresáře, kde je umístěn soubor s build skriptem), který obsahuje přeložené .java soubory, zdrojové a přeložené soubory vygenerovaných spojek a další pomocné soubory. Jako výsledek překladu je vytvořen .gar soubor (například soubor org_globus_services_MarIndex.gar pro službu IndexService), který obsahuje všechny potřebné informace pro deployment služby do kontejneru. Jméno souboru .gar se vygeneruje podle umístění .wsdd souboru. Tento soubor se poté příkazem "globus-deploy-gar org_globus_services_MarIndex.gar" umístí do kontejneru ("globus-undeploy-gar org_globus_services_MarIndex" pro vyjmutí služby z kontejneru) a po spuštění kontejneru ("globus-start-container -nosec") je služba přístupná na dané adrese.

Po spuštění kontejneru webových služeb se ve výpisu běžících služeb zobrazí mimo jiné tyto služby:

```
http://192.168.0.11:8080/wsrf/services/MarousekIndexFactoryService
http://192.168.0.11:8080/wsrf/services/MarousekPocasiFactoryService

http://192.168.0.11:8080/wsrf/services/MarousekIndexService
http://192.168.0.11:8080/wsrf/services/MarousekPocasiService
```

První dvě služby ve výpisu jsou factory a poslední dvě jsou IndexService a PocasiService.

Příloha B

Uživatelská příručka

Spuštění IndexService

Služba *IndexService* se při zachování adresářové struktury z přílohy A na straně 63 spouští příkazem:

```
java -classpath ./build/stubs/classes/:$CLASSPATH \
org.globus.clients.MarIndex.IndexClient \
http://192.168.0.11:8080/wsrf/services/MarousekIndexFactoryService
   IndexServiceEPR.txt
```

Zde se předpokládá, že v adresáři *build* jsou přeložené spojky služeb. Jako první parametr se službě předá URI factory pro *IndexService* a jako druhý název souboru, do kterého se uloží EPR na nově vytvořený zdroj a IndexService. Výpis klienta by měl vypadat takto:

```
Spusteni clienta.....
Vytvarim novy zdroj ..... hotovo
Zapisuji EPR do souboru IndexServiceEPR.txt ..... hotovo
```

V souboru *IndexServiceEPR.txt* by měl být nyní uložen EPR na IndexService a měl by vypadat takto:

```
<ns1:MarIndexResourceReference xsi:type="ns2:EndpointReferenceType"
     xmlns:ns1="http://www.globus.org/namespaces/MarousekIndexService"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:ns2="http://schemas.xmlsoap.org/ws/2004/03/addressing">
  <ns2:Address xsi:type="ns2:AttributedURI">
```

```
    http://192.168.0.11:8080/wsrf/services/MarousekIndexService
  </ns2:Address>
  <ns2:ReferenceProperties xsi:type="ns2:ReferencePropertiesType">
    <ns1:MarIndexResourceKey xmlns:ns1="http://www.globus.org/
        namespaces/MarIndexService_instance">
      5612943
    </ns1:MarIndexResourceKey>
  </ns2:ReferenceProperties>
  <ns2:ReferenceParameters xsi:type="ns2:ReferenceParametersType"/>
</ns1:MarIndexResourceReference>
```

Zde je vidět *URI IndexService* a *klíč* vytvořeného zdroje.

Spuštění PocasiService

Služba *PocasiService* se spouští příkazem:

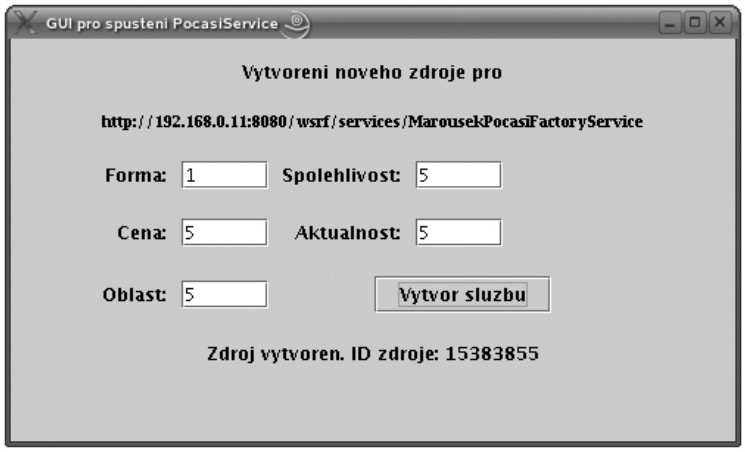
```
java -classpath ./build/stubs/classes/:$CLASSPATH \
org.globus.clients.MarService.PocasiClient \
http://192.168.0.11:8080/wsrf/services/MarousekPocasiFactoryService IndexServiceEPR.txt
```

Jako parametr je klientovi předáno URI factory pro *PocasiService* a název souboru, který obsahuje EPR na *IndexService*. Pokud vše proběhne v pořádku, je zobrazeno GUI pro spouštění (vytváření nových zdrojů) jednotlivých *PocasiService* (obrázek B.1 na straně 68). Po zadání statických hodnot nového zdroje a stisku tlačítka *Vytvor sluzbu* se klientem zavolá factory pro PocasiService, vytvoří se nový zdroj a je zobrazeno okénko, ve kterém uživatel spouští a zastavuje pravidelné změny dynamických dat pro příslušnou PocasiService (obrázek B.2 na straně 68). Po zavření tohoto okénka je zrušen daný zdroj a po chvilce je daná služba odregistrována u IndexService.

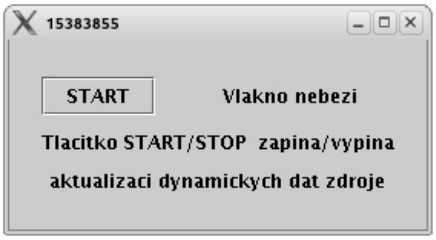
Připojení klienta k IndexService

Tento klient se spouští příkazem:

```
java -classpath ./build/stubs/classes/:$CLASSPATH \
org.globus.clients.MarClient.MarClient IndexServiceEPR.txt 1 2 9 2 7 3 9 3 9
```



Obrázek B.1: GUI pro vyváření zdrojů pro PocasiService



Obrázek B.2: GUI pro dynamickou aktualizaci zdrojů

Jako první parametr je klientovi předán název souboru, který obsahuje EPR na *IndexService*. Jako druhý parametr je očekáváno číslo 0 nebo 1, kterým si klient volí požadovaný druh výstupu (0-Text, 1-HTML). Poté následují MIN a MAX hodnoty těchto atributů - *Oblast, Aktualnost, Cena* a *Spolehlivost*. Tyto hodnoty musí být v intervalu 1-9 včetně. Pokud jsou parametry v pořádku a klient obdrží počasí v požadovaném městě, pak výstup klienta vypadá následovně:

```
Nacitam referenci na IndexService ze souboru IndexServiceEPR.txt ..... hotovo
Odesilam pozadavek IndexService
Zpozdeni: 2  Rychlost: 5 Vaha: 7 15383855:
Zpozdeni: 3  Rychlost: 8 Vaha: 11 25443492:
Zpozdeni: 6  Rychlost: 4 Vaha: 10 3312704:
Vyhrala sluzba na pozici: 1 s vahou: 7
Zjistuji pocasi ve meste Chomutov od sluzby s nejlepsi vahou ..... hotovo
Vysledek: <HTML><HEAD></HEAD><BODY>Soucasne pocasi ve meste Chomutov je:
    26 stupnu a Uragan</BODY></HTML>
```

Přehled chybových hlášení

Zde jsou popsány nejdůležitější chybová hlášení, která mají souvislost se službami.

MarIndexResource

Nenalezena sluzba pro UPDATE - zobrazí se ve chvíli, kde přijde *hello* zpráva do IndexService a uvedená PocasiService není nalezena v seznamu služeb

Vyprsel timeout sluzby s ID: - klient žádá seznam služeb, které vyhovují jeho požadavkům, a u této služby je čas posledního kontaktu větší než nastavený timeout. Služba je odebrána ze seznamu služeb v IndexService.

Nelze odebrat ze seznamu sluzbu s ID: - při pokusu o odebrání služby ze seznamu služeb v IndexService není tato služba v seznamu nalezena, a proto nemůže být odebrána

PocasiVlakno

ServiceException pri vytvareni MarIndexServicePortType - třída PocasiVlakno nemůže v konstruktoru vytvořit ukazatel na IndexService

ServiceException pri vytvareni MarServicePortType - třída PocasiVlakno nemůže v konstruktoru vytvořit ukazatel na PocasiService

Nelze se spojit s PocasiService - nelze se spojit s PocasiService při pravidelném posílání *hello* zprávy. Zdroj může být zrušen. Poté je PocasiService odregistrována z IndexService

Nelze odregistrovat sluzbu u IndexService - PocasiVlakno nemůže odregistrovat PocasiService u IndexService

Sluzba neni zaregistrovana u IndexService - PocasiVlakno se může spojit s IndexService, ale daná služba PocasiService není u IndexService zaregistrována

Chyba pri ruseni zdroje - nelze zrušit zdroj PocasiService buď po nenalezení služby u IndexService nebo po neúspěšném kontaktu IndexSevice za účelem poslání hello zprávy

Nelze se spojit s IndexService - nelze se spojit s IndexService při pravidelném poslání *hello* zprávy. IndexService může být zrušena. Poté je zrušena PocasiService

Evidenční list

Souhlasím s tím, aby moje diplomová práce byla půjčována k prezenčnímu studiu v Univerzitní knihovně ZČU v Plzni.

V Plzni dne 24. 8. 2005 František Maroušek

Uživatel stvrzuje svým čitelným podpisem, že tuto diplomovou práci použil ke studijním účelům a prohlašuje, že ji uvede mezi použitými prameny.

Jméno	**Fakulta / katedra**	**Datum**	**Podpis**