VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Vyhodnocovací server pro soutěže v programování

An Evaluation Server for Programming Contests

2012 Bc. Daniel Robenek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Daniel Robenek**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Vyhodnocovací server pro soutěže v programování
An Evaluation Server for Programming Contests

Zásady pro vypracování:

Cílem práce je implementovat webovou aplikaci, která bude sloužit jako vyhodnocovací server pro soutěže v programování.
Při návrhu a implementaci vyhodnocovacího serveru je třeba dbát zejména na robustnost celého řešení, odolnost proti útokům a možná bezpečnostní rizika, plynoucí z faktu, že soutěžící posílají na server kód, který je na něm následně spouštěn.

1. Prozkoumejte již existující vyhodnocovací servery a srovnejte je z hlediska poskytovaných možností.
2. Analyzujte možná bezpečnostní rizika, se kterými je třeba při návrhu vyhodnocovacího serveru počítat, a navrhněte způsoby, jak jim čelit.
3. Navrhněte a implementujte vlastní vyhodnocovací server. Server bude umožňovat:
a) Pořádání jednorázových soutěží (s daným časovým limitem).
b) Dlouhodobější spuštění, kdy je možno řešit určité úlohy mimo soutěže, např. pro účely tréningu, vyhodnocování úloh zadaných ve výuce apod.
c) Sledování statistik, aktuálního pořadí apod.
d) Archivaci veškerých zaslaných řešení.
e) Snadné přidávání dalších úloh.
f) Běh na Linuxu nebo jiném unixovém OS.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Zdeněk Sawa, Ph.D.**

Datum zadání: 18.11.2011
Datum odevzdání: 04.05.2012

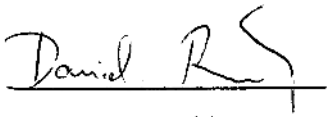
VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky OSTRAVA

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 1.5.2012



vlastnoruční podpis autora

Tímto bych rád poděkoval vedoucímu práce Ing. Zdeňku Sawovi, Ph.D. za cenné rady, připomínky a za metodické vedení při zpracování diplomové práce.

Abstrakt

Cílem této diplomové práce je prozkoumat veřejně dostupné vyhodnocovací servery pro soutěže v programování. Popsat jejich účel, strukturu, historii a technické aspekty. Ze získaných informací a údajů navrhnout a naimplementovat vlastní vyhodnocovací server. Systém rozdělit do modulů a u každého z nich popsat účel, problémy, návrh řešení těchto problémů a následně provést implementaci navrženého řešení.

Abstract

The goal of this thesis is to examine the publicly available servers for evaluating programming contests. To describe the purpose, structure, history and technical aspects. Then design and implement own server from collected information. Divide the system into modules and describe the purpose, problems and design of solutions of these problems and then to implement the proposed solution.

Klíčová slova

Vyhodnocovací server, online rozhodčí, soutěže v programování, kompilace, linux, ptrace, Java, C

Key words

Evaluation server, online judge, competition in the programming, compilation, linux, ptrace, Java, C

Seznam použitých symbolů a zkratek

ACM-ICPC	–	mezinárodní soutěž v týmovém programování
CRUD	–	zkratka pro akce vytvoření, čtení, úpravy a mazání
DI	–	technika pro nastavování závislostí v době běhu programu, podobný IoC
GIT	–	decentralizovaný systém pro správu verzí
IoC	–	návrhový vzor pro poskytování závislostí softwarovým komponentám
JDK	–	soubor nástrojů pro vývoj na platformě Java
JRE	–	běhové prostředí používané pro programovací jazyk Java
MVC	–	návrhový vzor využívaný webovými frameworky
PID	–	identifikační číslo procesu v operačním systému
SRS	–	dokument obsahující specifikaci softwarových požadavků
TCP	–	spojově orientovaný protokol transportní vrstvy

Obsah

Seznam tabulek

Seznam obrázků

1 Úvod

Během studia na vysoké škole jsem měl příležitost zúčastnit se soutěže CTU Open[1]. Jedná se o týmovou programátorskou soutěž, v níž se skupina jednoho až tří soutěžících snaží vyřešit zadané úlohy. Ti mají k dispozici jeden počítač, vytištěné zadání a pět hodin k dosažení co nejlepších výsledků.

Úlohy jsou většinou popsány příběhem s popisem problému. Při hlubším zamyšlení nad danou úlohou je zřejmá nutnost využít pro řešení efektivní algoritmus. Po jeho sepsání v programovacím jazyce se odesílá na server, který vyhodnotí správnost řešení.

V dalších ročnících studia na vysoké škole jsem se zapsal do předmětu seminář z programování. Ten má za cíl naučit studenty využívat k řešení úloh efektivní algoritmy a připravit je na soutěže v programování. Pro cvičení v tomto předmětu byl využit server Uva Onlie Judge[2], jež zajišťuje mimo jiné také zázemí pro kontrolu vypracovaných programů.

Vyvstal však požadavek vytvořit vlastní server, jež bude obsahovat specifické funkce a bude jej možno využít speciálně pro zpracování řešení v daném předmětu.

1.1 Cíl diplomové práce

V první části této diplomové práce budou popsány existující webové servery, které umožňují uživateli posoudit správnost řešení úlohy. U jednotlivých serverů budou prozkoumány jejich možnosti. Tedy počet a typ úloh, pořádání soutěží, komunita a další technické aspekty.

Následuje definice požadavků na implementaci vyhodnocovacího serveru. Budou detailně popsány její vlastnosti, které nejsou uvedeny v zadání diplomové práce.

Pro lepší přehlednost bude systém rozdělen na tři části. U každé z nich budou popsány základní problémy, jež má daný modul řešit. Následuje návrh daného modulu a popis jeho implementace ve vybraném programovacím jazyce.

Poslední část obsahuje instrukce k instalaci a spuštění vyhodnocovacího serveru na vlastním počítači.

1 Informace o soutěži CTU Open jsou dostupné na stránkách http://contest.felk.cvut.cz/.
2 Server UVa Online Judge je dostupný na adrese http://uva.onlinejudge.org/.

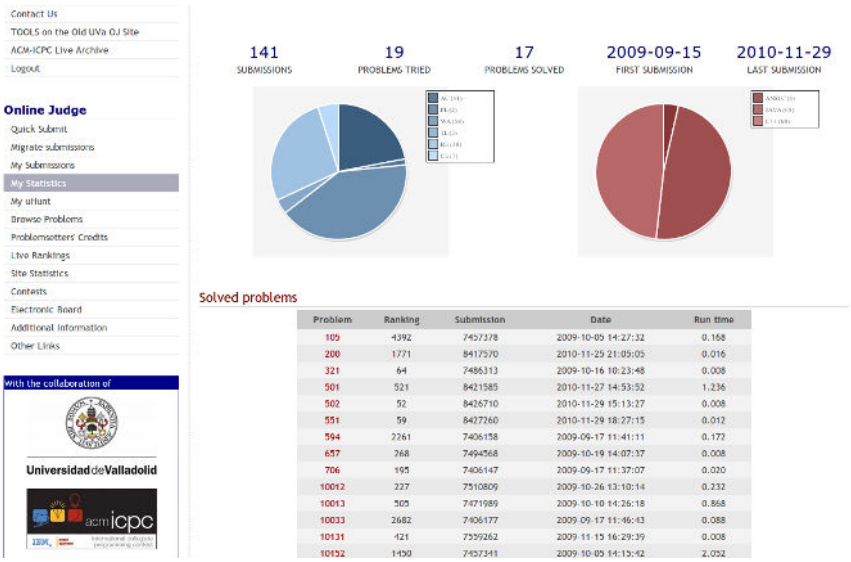
2 Existující vyhodnocovací servery pro soutěže v programování

V následující části jsou popsány nejznámější, veřejně přístupné vyhodnocovací servery pro soutěže v programování. Při jejich popisu je nahlíženo nejen na vlastnosti systému pro vyhodnocování úloh, ale i na vlastnosti s tím související. Například pořádané soutěže, typy úloh a jejich řešení a podobné.

2.1 UVa Onlie Judge

UVa Online Judge[3] je mezinárodní server zabývající se převážně hostováním systému pro ověřování správnosti odpovědí. Jeho historie sahá až do roku 1997, kdy od uživatelů přišly první správné odpovědi na zadané problémy. Od té doby odeslali uživatelé téměř deset miliónů odpovědí. Jedná se tedy o jeden z nejstarších serverů tohoto typu.

Databáze serveru Uva Online Judge obsahuje řádově tisíc problémů[21], lze zde nalézt také všechny problémy ze světových finále soutěže ACM-ICPC od roku 1990. Obsahuje sekci úloh, které jsou tématicky rozděleny vzhledem k algoritmu, jež by měl být při řešení použit. Po vyřešení úloh se lze podívat na svou statistiku, která obsahuje veškeré řešení odeslané uživatelem. Náhled této statistiky lze vidět na Obrázku 1.



Obrázek 1: Uživatelská statistika na serveru UVa Online Judge

Na stránky UVa Online Judge navazuje druhá část nazvaná ACM-ICPC Live Archive[4], ve které lze nalézt kromě výše zmíněných úloh z finále ACM-ICPC také regionální úlohy a to od roku 1988. Tato sekce stránek je zpřístupněna od roku 2003 a od té doby uživatelé odeslali téměř milión odpovědí.

3 Server UVa Onlie Judge je dostupný na adrese http://uva.onlinejudge.com/.
4 Archiv soutěží ACM-ICPC Live Archive je dostupný na adrese http://livearchive.onlinejudge.org/.

Server zveřejňuje jak nadcházející, tak i již proběhlé soutěže v programování, jako například rozehřívací kola na finále ACM-ICPC. Po ukončení soutěže bývají úlohy přidány do archivu, takže se může kdokoliv pokusit tyto úlohy vyřešit.

2.1.1 Technické aspekty serveru

Systém serveru UVa Online Judge je velice podobný systému používanému na soutěžích ACM-ICPC. Každá úloha může být řešena v programovacím jazyce ANSI C, Java 1.6, C++, nebo Pascal. Při kompilaci na ANSI C, nebo C++ je využit také přepínač pro optimalizaci kódu „-o2“ [22]. V sekci nápovědy se lze dočíst detaily ke specifickým vlastnostem jednotlivých programovacích jazyků při jejich použití na tomto serveru. Například se jedná o chybu, která znemožňuje systému odchytit běhovou výjimku při použití programovacího jazyku PASCAL[23]. Následně jsou zde uvedeny ukázky vypracovaných úloh v jednotlivých jazycích.

Po odeslání odpovědi na server je možno očekávat jednu z odpovědí uvedených v Tabulce 1[24].

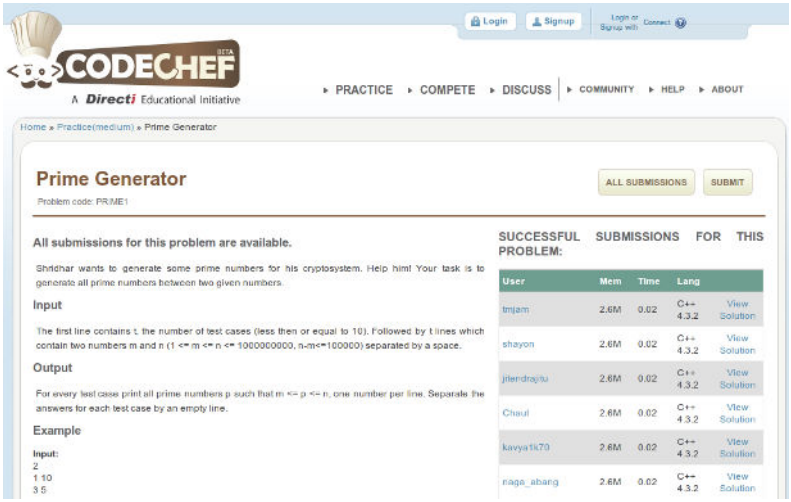
Název odpovědi	**Popis**
In Queue	zadaná úloha je ve frontě úloh ke zpracování, bude vyhodnocena jakmile to bude možné
Accepted	odpověď na úlohu je správná, běh programu skončil před časovým limitem a nepřekročil stanovený paměťový limit
Presentation Error	odpověď na úlohu je správná, ale není formátována dle požadavků
Wrong Answer	odpověď na úlohu není správná, respektive na vstupní data nebyl vydán správný výstup
Compile Error	program nešel korektně zkompilovat, chybový stav by měl být zaslán uživateli na e-mail
Runtime Error	v průběhu vykonávání programu vznikl neplatný stav, například dělení nulou, nebo program neskončil se stavovým kódem 0
Time Limit Exceeded	program potřebuje pro výpočet příliš mnoho času
Memory Limit Exceeded	program se snažil alokovat více paměti, než bylo povoleno v zadání úlohy
Output Limit Exceeded	výstup programu je neočekávaně velký, například kvůli nekonečné smyčce s výpisem
Submission Error	odeslání odpovědi nebylo úspěšné, například nastalo neočekávané ukončení odesílání programu
Restricted Function	program se snaží využít nepovolenou funkcionalitu systému, jako například pokus o otevření souboru, pokus o navázání síťového spojení a podobně
Can't Be Judged	některé problémy dosud nemají testovací data, proto nemůže být přijat požadavek na otestování správnosti odpovědi

Tabulka 1: Možné odpovědi serveru UVA Online Judge

2.2 Codechef

Codechef[5] je indický výukový server pro trénink algoritmických problémů a pořádání soutěží. Tento server využívá již přes 25 000 uživatelů[5], jež poslali okolo 800 000 odpovědí[6]. Kontrola správných odpovědí však není jeho jediná vlastnost, která stojí za zmínku. Jako celek ho lze rozdělit do tří sekcí.

Sekce trénink obsahuje přes 700 úloh rozdělených do skupin dle obtížnosti. U každé úlohy lze vidět počet úspěšných řešitelů, úspěšnost řešení a také kód správných řešení. Obtížnější sekce obsahují úlohy přes dva roky staré, avšak stále nevyřešené. Pod úlohou se nacházejí komentáře s dotazy. Ty však většinou zůstávají nezodpovězeny. Na Obrázku 2 lze vidět obrazovku se zadáním úlohy.



Obrázek 2: Zadání úlohy "Prime Generator" na serveru CodeChef

V následující sekci lze nalézt soutěže, jež jsou uspořádávány jednak samotným serverem codechef, tak i externími organizacemi. Soutěže jsou buď zcela otevřené, nebo pouze pro studenty. Po ukončení soutěže je umožněno nahlédnout na správné odpovědi od soutěžících a u některých soutěží je také popsán způsob správného řešení úlohy.

Poslední část tohoto serveru je zaměřena na komunitu. Ta zde může probírat svá řešení na fóru. Jsou zde také sekce pro pravidelné soutěže, takže případné nejasnosti či nápady mohou být řešeny právě zde. Pro nově příchozí je připravena wiki, na které je možno se dozvědět veškeré potřebné informace pro práci se serverem. Server také spravuje blog, který je zdrojem informací o novinkách týkajících se serveru. Jsou zde zveřejňovány výsledky soutěží, úpravy serveru a další užitečné informace.

5 Server Codechef je dostupný na adrese http://www.codechef.com/.
6 Tato informace byla odhadnuta z ID aktuálně zaslané odpovědi.

2.2.1 Technické aspekty serveru

Server codchef se snaží být k uživatelům co nejbenevolentnější. Pro vypracování příkladu lze použít jeden z více než 40 programovacích jazyků. Přesto v odpovědích dominují jazyky C, C++ a Java[7]. Po odeslání odpovědi na server je možno očekávat jednu z následujících odpovědí uvedených v Tabulce 2.

Název odpovědi	**Popis**
Accepted	zaslaný program má předpokládané chování, tedy na daný vstup vydá předpokládaný výstup v zadaném čase.
Time limit exceeded	zaslaný program možná funguje správně, ale potřebný čas pro zpracování je příliš vysoký
Wrong answer	zaslaný program nemá předpokládané chování, respektive na vstupní data nevydal správnou odpověď
Runtime error	při běhu programu nastaly neplatné stavy, jako například pokus o dělení nulou, nebo požadavek na alokaci příliš mnoho paměti
Compilation error	zaslaný kód nešel zkompilovat, podrobnosti o této chybě jsou uživateli k dispozici

Tabulka 2: Možné odpovědi serveru CodeChef

V případě běhové chyby může být uživatel informován o jeho důvodu. V nápovědě jsou vysvětleny chybové stavy jako například *SIGSEGV*, *SIGABRT*, nebo *SIGFPE*. U jednotlivých programovacích jazyků je také zohledněna jejich režie. Například programovací jazyk Java má na zpracování vstupních dat dvojnásobné množství času. U jazyků jako Ruby, Python, nebo PHP je tento čas dokonce trojnásobný[4].

Server codechef navíc zavádí skóre. Jedná se o nástroj, který umožňuje ohodnotit částečné, nebo přibližně správné řešení daného problému. Pokud daný program řeší zadání s určitou chybou, nebo část neřeší vůbec, přičte se k výslednému hodnocení penalizace. Při zveřejňování úspěšných řešení je pak zohledněna nejen rychlost s jakou program daný problém vyřeší, ale i penalizace jež uživatel dostane.

2.3 Light OJ

Light OJ[8] je teprve rok starý server, který se snaží, ostatně jako předchozí servery, zajistit možnost procvičit se v logickém myšlení a schopnosti efektivně řešit složité algoritmické úlohy.

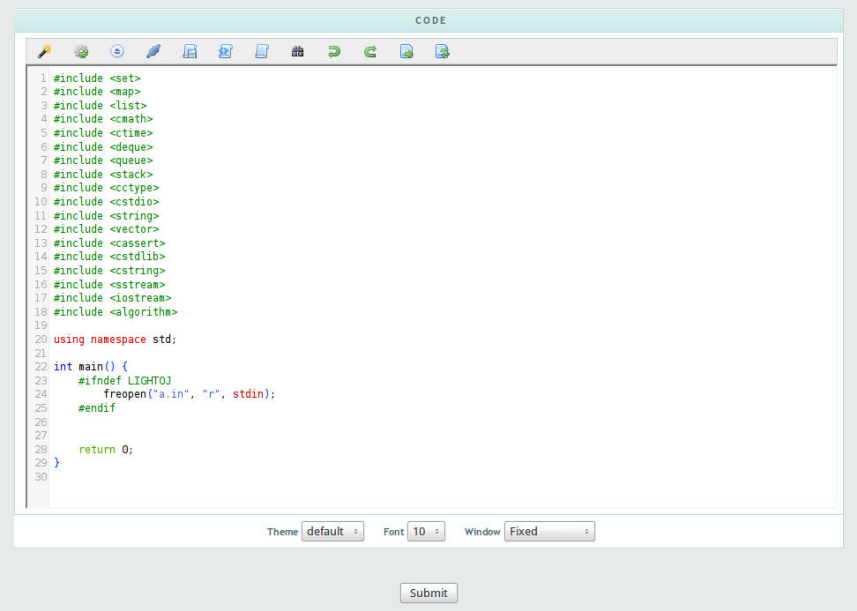
Po tomto krátkém čase svého působení má přibližně 1500 uživatelů, jež si mohou vybrat k procvičení z více než 400 dostupných úloh. Většina uživatelů pochází z Bangladéše, Indie a Číny [7].

K úlohám lze přistupovat jednak sekvenčním způsobem, seřazeny dle data přidání, nebo přes kategorie do nichž jsou úlohy přiřazeny. Kategorie jsou tvořeny dle algoritmu, jež je potřeba využít ke správnému vyřešení úlohy.

7 Tato informace byla získána procházením seznamu řešení několika desítek úloh.
8 Server Light OJ je dostupný na adrese http://www.lightoj.com/.

Po registraci a přihlášení do systému je na úvodní obrazovce popsán postup, jak začít server používat. Pro nováčky je zde sekce jednodušších problémů, u kterých mohou začít. Následně jsou nasměrováni do seznamu úloh rozdělených podle kategorií.

Za zmínku stojí integrovaný webový editor s kompilerem, jež může nahradit běžné vývojové nástroje. Tato pomůcka je velice užitečná hlavně pro začátečníky. Po načtení stránky se zobrazí standardní okno s předpřipraveným kódem, do něž se připisuje kód. Pokud uživatel vloží číslo úlohy, může si po napsání programu a jeho úspěšné kompilaci otestovat jeho správnou funkčnost oproti ukázkovému výstupu. Ukázku tohoto kompileru lze vidět na Obrázku 3.



Obrázek 3: Integrovaný editor na serveru LightOJ

Server obsahuje také výukovou sekci, ve které se nacházejí algoritmy a jejich popis s ukázkami praktických použití. Nutno podotknout, že prozatím obsahuje pouze 6 článků.

I tento server obsahuje komunitní sekci, ve které uživatelé mohou diskutovat nad uvedenými problémy. Fórum také obsahuje sekci, ve které lze probírat řešení úloh z jiných soutěžních serverů. V případě zájmu může uživatel přidat úlohu k řešení. Pokud již uživatel vyřešil alespoň 25 úloh, může na tomto serveru také uspořádat soutěž.

2.3.1 Technické aspekty serveru

Na serveru Light OJ je možno k vyřešení úlohy použít jeden z pěti programovacích jazyků, a to C, C++, Java, Pascal a Python. Všechny tyto programovací jazyky je možno použít také ve webovém kompileru.

Význam veškerých typů odpovědí serveru byl již uveden v Tabulce 1. Proto je zde pouze výpis jejich názvů, tedy *accepted*, *wrong answer*, *compile error*, *presentation error*, *runtime error*, *time limit exceeded*, *memory limit exceeded*, *output limit exceeded* a *restricted function*.

2.4 Sphere online judge

Kolem roku 2004 byl spuštěn nový polský server se zkratkou SPOJ[9]. Původní záměr tvůrců tohoto serveru bylo využít tento systém jako pomůcku při výuce. Mnoho polských univerzit již několik let využívalo serverů tohoto typu, avšak záměr tvůrců bylo vytvořit server s větším rozsahem funkcionalit.

Jejich snaha byla zaměřena na uživatele, jež nebyli spokojeni s již existujícími systémy a nabídnout jim alternativu. Systém je tedy primárně zaměřen na učitele a studenty univerzit a na komunitu programátorů, jež mají zálibu v řešení složitých algoritmických programátorských úloh[19].

Tento server nabízí v současné době k řešení téměř 3 000 veřejných problémů řešitelných pomocí 46 programovacích jazyků. Za dobu své existence tento server využilo téměř 140 000 uživatelů, kteří odeslali přes 6 700 000 odpovědí k vyhodnocení. Mimo jiné je tento server možno využít k pořádání vlastních soutěží, kterých již bylo přes 2 000.

2.5 Project euler

Server Project Euler[10] se zásadně liší od ostatních zmiňovaných serverů. Úlohy, které lze na tomto serveru nalézt, jsou více zaměřeny na dobrou znalost matematiky, než na samotné programování. Důkazem toho jsou již první úlohy. Součet čísel dělitelných 3 nebo 5 a zároveň menších než 1000, součet sudých čísel Fibonnaciho posloupnosti menších než milión, nebo největší prvočíselný dělitel zadaného čísla. Další úlohy pak stupňují náročnost na vyřešení.

Rozdíl oproti ostatním serverům je také ve stylu kontroly výsledku. Zde se neposílá soubor se zdrojovým kódem, ale jednorázová, číselná odpověď na danou otázku. Toto řešení má své výhody i nevýhody. Výhody jsou například jednoduchost implementace serveru, kontroly správnosti odpovědi, nebo příprava úlohy. Jako hlavní nevýhodu spatřuji v nemožnosti přesnější kontroly časové náročnosti výsledného programu.

Projekt se snaží zaujmout uživatele možností získat ocenění za úspěšně vyřešené úlohy. Například za první tři vyřešené problémy, dvanáct problémů mající ID Fibonacciho posloupnosti, první uživatel který správně vyřeší úlohu a podobné. Dle aktuálních statistik je zde zaregistrováno přes 200 000 uživatelů, kteří odeslali téměř 3 500 000 správných odpovědí[11].

9 Server Sphere Online Judge je dostupný na adrese http://www.spoj.pl/.
10 Server Project Euler je dostupný na adrese http://projecteuler.net/.

3 Zadání systému

Nyní nastává chvíle specifikovat vlastnosti, jež má implementace vyhodnocovacího serveru zahrnovat. Pro tento požadavek existuje speciální dokument. Jedná se o dokument SRS, tedy o specifikaci softwarových požadavků.

Tento dokument však neobsahuje pouze funkční požadavky systému. Lze v něm nalézt účel systému, seznam uživatelů, požadavky na uživatelské rozhraní, hardware, software a podobně. Díky tomu je jasně specifikováno prostředí, v němž výsledný systém poběží.

Vzhledem k rozsahu je SRS vložen jako příloha. Dokument SRS byl vypracován do šablony[11], jež byla upravena pro specifický průběh vývoje software.

11 Šablonu lze získat například na webové adrese http://cs.gmu.edu/~kdobolyi/cs421/srs_template.doc.

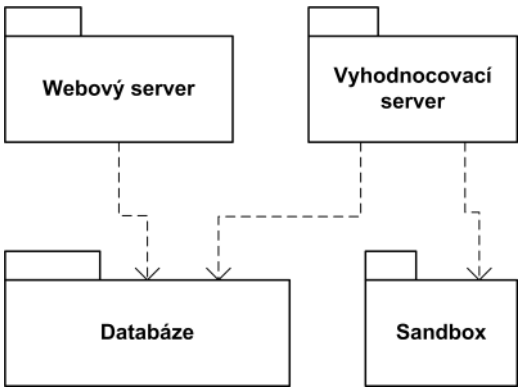
4 Rozdělení systému

Pro lepší správu kódu a pochopení celého systému je projekt rozdělen do několika celků – modulů, které samostatně obstarávají přidělené funkce. Při návrhu jak systém rozdělit byly brány v úvahu faktory jako závislost na ostatních částech systému, požadovaná funkcionalita modulu a smysl samotného oddělení od zbytku systému.

Po zvážení těchto faktorů jsem tento projekt rozdělil do následujících částí:

- webový server
- vyhodnocovací server
- sandbox
- databáze

Propojení mezi moduly je omezeno na minimum, aby nedocházelo k nadměrnému ovlivňování jednotlivých částí v případě změny jejich vnitřní implementace. Vzájemné závislosti lze vidět na Obrázku 4.



Obrázek 4: Rozdělení systému do modulů

Pro lepší porozumění tohoto rozdělení budou stručně popsány úkoly jednotlivých modulů v rámci vyhodnocovacího serveru.

Webový server je jediným prostředkem, se kterým může běžný uživatel komunikovat. Pomocí webového prohlížeče a svých přihlašovacích údajů se v systému uživatel autentizuje a následně může provádět úkony specifikované v zadání systému. Tento modul musí umět také správně validovat vstupní data a starat se o autorizaci požadavků. Samotný webový server by neměl být s vyhodnocovacím serverem jakkoliv přímo propojen. Pro lepší odezvu systému je však zavedena notifikace[12]. Nutnost propojení webového serveru s databází je zřejmá. Veškerá dynamická data jsou uložena v databázi a při zpracování požadavku přes webový prohlížeč se webový server na tyto data dotazuje databáze.

Vyhodnocovací server má v základu jedinou funkci, a to vybírat nevyřešené úlohy

12 O notifikaci pojednávají kapitoly 6.2.2 a 6.5.7.

z databáze a vyhodnotit jejich správnost. Přestože je toto jediný požadavek, je tento modul dosti rozsáhlý. Napojení na databázi je zřejmé, avšak v případě potřeby by šla tato závislost převést na jiný modul – prostředníka. Z databáze se získávají potřebné úlohy a po vyhodnocení se zapíše výsledek. Spojení se sandboxem zajišťuje vyhodnocovacímu serveru nezávislost vzhledem k použitým programovacím jazykům v soutěžích. V případě potřeby je přidání jiného programovacího jazyka možno bez zásahu do zdrojového kódu vyhodnocovacího serveru. Změní se pouze hodnoty v konfiguraci serveru a přidá se v nich také závislost na jiný sandbox.

Sandbox samotný má za úkol zabezpečit spuštění programu soutěžícího. Jelikož tento program může provádět zakázané činnosti je úlohou sandboxu tyto úkony odhalit a běh programu ukončit. Mimo jiné má za úkol zjistit čas potřebný pro samotný běh programu a jeho spotřebu paměti. Po ukončení jeho běhu tyto skutečnosti předá zpět vyhodnocovacímu serveru.

Databázi netřeba představovat. Jedná se o úložiště veškerých dynamických dat serveru, jež jsou při jeho běhu používány. V případě potřeby zálohování je tedy nutno uložit pouze data z databáze.

5 Webový server

5.1 Úlohy modulu

Modul webového serveru se má starat o veškerou interaktivní komunikaci s klientem, tedy s webovým prohlížečem. V tomto modulu má být integrována logika, jež je nutná pro pohodlnou práci uživatele s vyhodnocovacím serverem. Nestará se však o ukládání dat, jež má za úkol databáze a o samotné vyhodnocení úlohy. To má na starosti modul vyhodnocovacího serveru.

Veškerá funkcionalita, kterou má tento modul na starosti, je sepsána v dokumentu SRS. Jsou zde uvedeny požadavky na správu uživatelů, jejich skupin, úlohy, soutěže a podobné. V následujících kapitolách budou tyto požadavky rozepsány detailněji.

5.1.1 Funkce webového serveru z pohledu administrátora

Následující seznam funkcí je potřebný pro uživatele, jež se přihlásí do systému jako administrátor. Následující funkce by neměly být přístupny pro uživatele, jež se přihlašují do systému jako soutěžící, případně nejsou přihlášeni vůbec.

Základem celého systému je správa úloh, které mohou uživatelé řešit. Ta kromě základních údajů jako název, popis, nebo kategorii musí obsahovat také program, jež dokáže vyhodnotit správnost výstupu testovaného programu. Vstupní data musí obsahovat každá úloha. Výstupní data jsou nepovinná a jsou zde pouze jako pomocná data pro vyhodnocovací program. Pokud tedy nejsou nutná, například pokud lze správnost výsledku ověřit výpočtem, výstupní data nejsou zadána.

Každé úloze lze přiřadit kategorii, například typ algoritmu, pomocí něhož se má daná úloha řešit, nebo soutěž, na níž byla úloha prvně použita.

Následně je nutno mít možnost spravovat soutěže, jež reprezentují množinu úloh k řešení. Soutěž může, ale nemusí být časově ohraničena. Pokud není oboustranně časově ohraničena, nepovažuje se za soutěž v pravém slova smyslu, ale spíše za trénink. Pokud však je časově ohraničena, je možno vyplnit parametr, jež reprezentuje časovou penalizaci za odeslání špatné odpovědi. Do soutěží se přiřazují týmy reprezentující uživatele přiřazené do soutěže. U každé soutěže vidí administrátor statistiky, jež určují pořadí jednotlivých soutěžících.

Tyto týmy jsou seskupení uživatelů, jež mají za úkol jejich snadnější správu. Před vytvořením soutěže se daní uživatelé přiřadí do stejného týmu a tento tým se přiřadí k soutěži. Následně není problém tento tým dále využívat během dalších soutěží.

Samotný uživatel je reprezentován svým uživatelským jménem a heslem. Další údaje jsou nepovinné a jsou pouze pro administrátora. Pro zákaz přihlášení do systému je možno uživatele dočasně deaktivovat.

Vstupní a výstupní data jsou u většiny úloh rozdílná. Je však vhodné mít možnost využít program pro vyhodnocení správnosti výsledku v systému opakovaně. Například u mnoha úloh stačí

pro ověření správnosti porovnat výstup testovaného programu s očekávaným výsledkem.

Následují funkce jako správa administrátorů, změna hesla a odhlášení ze systému.

5.1.2 Funkce webového serveru z pohledu soutěžícího

Soutěžící po přihlášení do systému by měl mít možnost vybrat si ze soutěží, v nichž chce řešit úlohu. Po jejím výběru by se měly zobrazit její informace a úlohy, které soutěž obsahuje.

Výběrem úlohy se uživateli zobrazí její informace, tedy název, časový limit, paměťový limit, limit na zdrojový kód, popis, ukázkový vstup a ukázkový výstup.

Po vyřešení úlohy ji uživatel přes webový formulář odešle k vyhodnocení. Následně uvidí svá odeslaná řešení a statistiku o jejich správnosti. Jako další připadá v úvahu statistika s uživateli a jejich pořadím. Ta se však v nastavený čas před skončením soutěže přestane aktualizovat.

Následují poslední dvě akce, které může uživatel provést, a to změna hesla a odhlášení.

5.2 Použité technologie

Při výběru programovacího jazyka pro implementaci jsem hledal ideální řešení, jež by spojovalo programovací jazyk s dobrým webovým frameworkem. Navzdory špatné pověsti jsem původně chtěl využít programovací jazyk PHP spolu s výtečným Nette frameworkem. Od této kombinace jsem však upustil a využil jsem programovací jazyk Java.

Pro jednodušší a rychlejší vývoj webové aplikace jsem využil Play framework ve verzi 1.2.4[13]. Jedná se o framework založený na návrhovém vzoru MVC, jež rozdělí aplikaci na základní tři oblasti. Na *model*, *view* a *controller*. *Model* obsahuje logiku aplikace a stará se o funkce zajišťující její spolehlivý běh. *View*, tedy jakási šablona, obsahuje definici vzhledu stránky. *Controller* navzájem propojuje předchozí dvě části[20]. Toto je však pouze velmi zjednodušený pohled na tento návrhový vzor.

Jako nástroj pro objektově-relační mapování využívá Play framework nástroje Hibernate. Nad ním je postavena nadstavba, jež mění jeho strukturu na návrhový vzor *active record*[1].

5.2.1 Moduly Play frameworku

Play framework nabízí možnost využití připravených modulů rozšiřujících samotnou funkcionalitu tohoto frameworku. Během vývoje webového serveru byly některé z nich využity, případně modifikovány pro potřebu vývoje.

Jeden ze základních modulů, jež lze použít, má název *Secure*. Tento modul, jak již název napovídá, se stará o přihlášení uživatele do systému a následnou autorizaci jednotlivých operací. Použití lze nalézt téměř ve všech controllerech, kde jsou nad třídou anotace `@With(controllers.Secure.class)` a `@Check`. První z této dvojice zajišťuje aplikaci autorizace uživatele a druhá definuje roli, v níž musí uživatel být.

13 Stránky Play frameworku lze nalézt na adrese http://www.playframework.org/.

Další využitý modul má název *CRUD*. Díky němu je mnohem snazší vytvářet standardní akce pro vytváření, editaci a mazání jednotlivých objektů. Tento modul je velmi ceněný ve většině webových frameworcích.

Pro zobrazení navigace byl využit modul *navigation*. U tohoto modulu je nutno definovat konfigurační soubor *conf/navigation.yml*, do kterého je zapsána množina položek v menu a jejich propojení. Pro správné zobrazení tohoto menu však bylo nutno lehce upravit soubor pro jeho vykreslení.

Poslední použitý modul má název *table*. Jak už název napovídá, jedná se o snadné vykreslování tabulek.

5.3 Datový model

Webový server využívá řadu entit, jež reprezentují uložitelná data. Třídní diagram těchto entit lze nalézt na Obrázku 17. Kvůli přehlednosti bylo odebráno několik propojení, hlavně s entitou *Admin*.

Všechny tyto entity dědí od třídy `play.db.jpa.Model`, jež implementuje funkce pro práci s entitou jako s modelem typu *AcriveRecord*. Navíc jsou proměnným těchto modelů přiřazeny anotace umožňující provést kontrolu entity před samotným uložením. Díky tomu framework dokáže snadno vyhodnotit chybu v entitě a informovat o této skutečnosti uživatele[25].

Entity zobrazené v diagramu se kryjí se zadáním a jejich názvy jsou samopopisné, proto nebudou funkce jednotlivých části detailněji popisovány.

5.4 Návrh a implementace

Rozdělení webového serveru do balíčků je uvedeno v Tabulce 3.

Název balíčku	**Funkce**
controllers	základní controllery, například pro přihlášení, nebo změnu hesla
controllers.admin	controllery využívající administrátor
controllers.admin.with	pomocné třídy využívané controllery administrátora
controllers.contestant	controllery pro stránky soutěžících
controllers.contestant.with	pomocné třídy využívané controllery soutěžícího
models	modely
services	v tomto balíčku je pouze služba pro notifikaci serveru
services.competition	služby využívané controllerem spravujícím závody
services.task	služby využívající controllerem spravujícím úlohy

Tabulka 3: Balíčky webového serveru

Pro implementaci byly využity doporučené postupy týkající se Play frameworku. Názvy tříd v balíčcích *controllers* reprezentují jednotlivé položky menu.

5.4.1 Výpočet statistik

Během soutěže je možno zobrazovat pořadí jednotlivých uživatelů. Je však potřeba rozlišovat soutěže a trénink. Jelikož trénink není přesně časově ohraničen, nelze pro něho počítat časové penalizace.

Samotný algoritmus pro výpočet statistiky, respektive pořadí uživatelů, je implementován ve třídě `services.competition.ContestantsStatistics`. Ta ve své veřejné metodě `getStatistics` nejprve kontroluje, zdali se jedná o soutěž a vyhodnotí datum, do něhož se mají statistiky počítat. Tato akce je nutná, jelikož soutěžícím se musí v určitý okamžik přestat aktualizovat statistika.

Následuje cyklus přes všechny uživatele. U každého uživatele je zjištěn počet vyřešených úloh, počet neúspěšných pokusů u těchto úloh a čas jejich vyřešení. Následně je vypočtena penalizace. V případě soutěže se vypočte jako

$$p = \sum_{i=0}^{r} \left(t_i + 60000 \,.\, CP \,.\, isc_i \right)$$

kde p je celková penalizace, r je celkový počet vyřešených úloh, t je čas od počátku do vyřešení úlohy v milisekundách, CP je časová penalizace za špatné řešení v minutách a isc je počet špatných řešení dané úlohy.

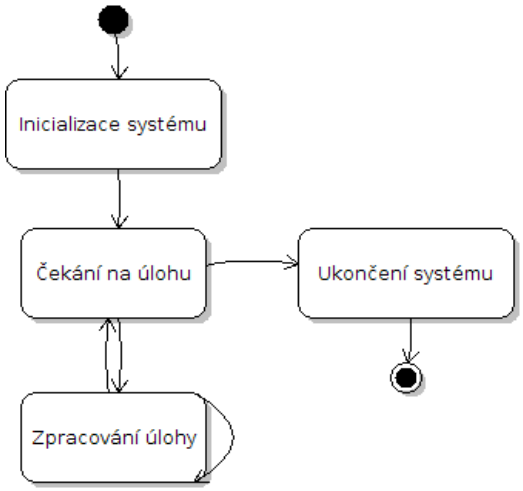
Pokud se však jedná pouze o trénink, je penalizace vypočtena čistě jako počet špatných řešení u vyřešených úloh.

6 Vyhodnocovací server

6.1 Úlohy modulu

Modul vyhodnocovacího serveru má za úkol jedinou činnost a to vyhodnotit správnost odpovědi na úlohu.

Ač se jedná na první pohled o jednoduchou úlohu, je nutno si ji rozebrat detailněji. Velmi hrubý pohled na činnost systému lze vidět na Obrázku 5. Z něj je patrné, že po inicializaci systému přejde server do neaktivního stavu. V tomto stavu server čeká na vnější podnět, jež indikuje přítomnost nové, nevyřešené úlohy. Následně tuto úlohu zpracuje a buď zpracuje další, nebo přejde do neaktivního stavu.



Obrázek 5: Stavový diagram vyhodnocovacího serveru

Vyhodnocení samotné úlohy je základní úkon tohoto modulu, avšak při detailnějším zamyšlení se lze dostat k následujícím problémům, jež by měl tento modul řešit:

- komunikace se zdrojem dat
- efektivní způsob zjištění přítomnosti nevyřešené úlohy
- správa potřebných vstupních souborů úlohy
- kompilace zdrojového kódu řešení úlohy
- propojení/spuštění sandboxu
- kontrola správnosti výsledku
- odstranění již nepotřebných souborů

Následující návrh modulu vyhodnocovacího serveru se tedy bude snažit co nejlépe vyřešit problémy spojené s výše uvedenými požadavky.

Vyhodnocovací server by měl být lehce rozšiřitelný. Pokud vznikne požadavek na použití různých programů na kontrolu správnosti výsledku, nebo použití různých sandboxů, mělo by to být možno bez nutnosti zásahů do kódu.

Dalším požadavkem je dostatečná rychlost běhu, respektive paralelizmus při vyhodnocování řešení úloh. Samotné vyhodnocování je tedy vhodné provádět v samostatném vlákně pro každé řešení. Běžně používané soubory pro vyhodnocování jednotlivých úloh by měly být zachovány pro další použití a neměly by se mazat.

6.2 Návrh modulu

Pro funkčnost tohoto modulu je nutno navrhnout řešení dříve zmíněných problémů. Při pohledu na modul vyhodnocovacího serveru jako na celek je možno zjistit celkovou provázanost systému. Na Obrázku 18 lze detailněji spatřit jeho průběh.

Tento diagram zatím nebere v potaz paralelní zpracování. Po inicializaci serveru, tedy například po navázaní spojení se zdrojem dat a vytvoření ostatních datových spojení, je prováděn základní cyklus modulu.

Podmínkou pro ukončení cyklu je vznik požadavku na ukončení vyhodnocovacího serveru. Pokud tato situace nenastane, je zkontrolována přítomnost nevyhodnoceného řešení úlohy. Pokud nevyhodnocené řešení úlohy neexistuje, je server pozastaven a čeká na zprávu o její přítomnosti.

Pokud nevyhodnocené řešení existuje, je zahájen proces jeho vyhodnocení. Nejprve je toto řešení úlohy označeno jako rozpracované a jsou získány jeho detaily. Pak následuje získání zdrojových kódů řešení a vytvoření souboru, do něž je zdrojový kód zapsán.

Je provedena kompilace zdrojového kódu do programu a vytvoří se vstupní soubor pro danou úlohu. Dále je program spuštěn v sandboxu patřícímu danému programovacímu jazyku. Ten vytvoří soubor s daty, jež daný program zpracoval. Tato data jsou předána vyhodnocovacímu programu. Ten na výstup vydá odpověď, zdali řešící program pracuje správně, nebo ne.

Následuje odstranění dále nepotřebných souborů a zapsání informace o řešení úlohy. V případě vzniku chyby při kompilaci programu, nebo při jeho spuštění v sandboxu je další průběh vyhodnocování pozastaven a provede se pouze odstranění nepotřebných souborů a zapsání výsledku.

Vyhodnocovací server dále pokračuje v cyklu.

6.2.1 Komunikace se zdrojem dat

Jako zdroj dat by mělo obecně existovat rozhraní, jehož implementace by byla nahraditelná. Za základní zdroj dat by měla být považována databáze. Modul vyhodnocovacího serveru pro svou základní funkčnost potřebuje mít možnost atomicky získat nevyhodnocené řešení úlohy a zapsat výsledky jeho vyhodnocení.

Získání nevyhodnoceného řešení úlohy zajistí data, jež jsou potřeba pro vyhodnocení.

Je nutno získat nejen zdrojový kód řešení, ale i typ programovacího jazyka, informace o úloze, limity pro řešení a další. Jelikož by jedno řešení úlohy nemělo být vyhodnocováno více procesy, je nutno jej označit jako aktuálně zpracovávanou. Toto označení je provedeno časovým razítkem záznamu, jež označuje expiraci jejího zámku. V případě překročení tohoto času je velice pravděpodobné, že úloha z nějakých příčin nebyla vyhodnocena a jiný proces se jí může pokusit vyhodnotit.

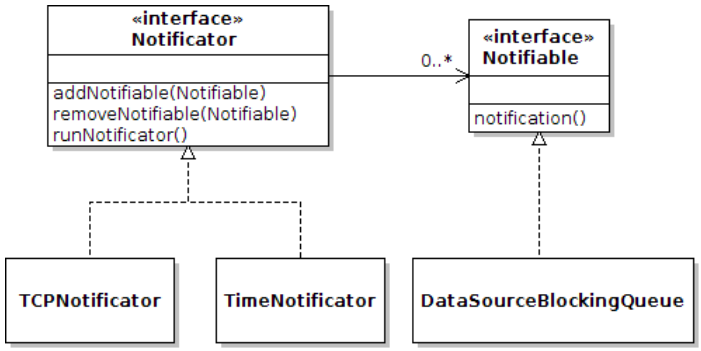
Zapsání výsledku úlohy je druhá z nutných funkcionalit. Před samotným zapsáním výsledku je nutno zkontrolovat časový zámek. Při jeho překročení nutno v zápise nepokračovat.

6.2.2 Zjištění přítomnosti nevyřešené úlohy

Při běhu vyhodnocovacího serveru nastává situace, kdy neexistuje nevyhodnocené řešení úlohy. Po zjištění této skutečnosti má server několik možností, jak se zachová.

Jedna z možností je v cyklu kontrolovat přítomnost nevyhodnoceného řešení. Toto je sice nejlehčí, avšak zdaleka ne nejlepší řešení. Pokud by bylo použito, znamenalo by značnou zátěž na samotný systém.

Z praktického hlediska je vhodné vlákno uspat a pro probuzení využít návrhový vzor *observer*[8], jak je ukázáno na Obrázku 6. Implementace pak nebude vázána na jeden konkrétní způsob probuzení hlavního vlákna. Po jeho uspání může jakýkoliv objekt – notifiátor toto vlákno probudit a to bude pokračovat ve zpracovávání řešení.



Obrázek 6: Použití návrhového vzoru observer u notifikátorů

Základní notifikátor, jež server obsahuje, je založen na časovači. Ten lze nastavit na libovolný interval. Po jeho spuštění a uplynutí daného času je vyslána notifikace všem příjemcům. Tento proces se dále opakuje.

Notifikace na základě časového intervalu je sice spolehlivý prostředek na pravidelnou kontrolu přítomnosti nevyhodnoceného řešení úlohy, avšak nastavení časového intervalu je kompromis mezi zatížením systému a rychlostí zpracování řešení úlohy.

Jako alternativu, respektive doplněk k časové notifikaci, je vhodné přidat také notifikaci založenou na příchozí zprávě. V konkrétním případě je jako zpráva brán pokus o navázání TCP

spojení se serverem. V případě výskytu tohoto připojení je vyvolána notifikace a spojení je ukončeno. Tímto způsobem je docíleno okamžité reakce serveru na přijetí řešení úlohy. Časový notifikátor je pak zde pouze jako záloha.

6.2.3 Správa souborů

Při vyhodnocení úlohy je potřeba vytvářet a mazat mnoho souborů. Při jediném vyhodnocení řešení úlohy se počet souborů, jež je nutné vytvořit, pohybuje okolo šesti. Navíc je při každém nestandardním ukončení vyhodnocení úlohy počet souborů k následnému odstranění jiný. Je tedy vhodné vytvořit systém správy dočasných souborů.

Tento nástroj by měl mít následující funkce:

- vytvořit soubor ze zdroje dat vyhodnocovacího serveru, například z databáze
- vytvořit nový soubor, respektive zablokovat určitý název souboru bez jeho vytvoření
- přiřadit souboru ID skupiny – tag
- odstranit daný soubor
- odstranit všechny soubory s daným ID skupiny

Veškeré tyto funkce by měly být zabezpečeny, aby je bylo možno využívat z více vláken. Běžnou operací je vytvoření souboru ze zdroje dat, tedy databáze. V databázi je uložen název souboru a jeho data. Vytvoří se nový, dosud nepoužitý soubor. Tento soubor je pak naplněn daty. Nastává také potřeba pouze zablokovat název souboru, tedy fyzicky soubor nevytvářet. Do paměti se uloží nový, náhodný název souboru, kde se bude kontrolovat na duplicity.

Při vyhodnocování řešení úlohy nastává mnoho situací, ve kterých by bylo velmi obtížné správně odstranit všechny soubory. Jedná se například o stav při zachycení výjimek, nebo při složitých větveních programu. Je tedy vhodné mít možnost si při vytváření souboru definovat ID skupiny, respektive tag. Podle tohoto tagu lze pak se soubory dále hromadně pracovat.

Po ukončení potřeby pracovat se soubory je vhodné tyto soubory odstranit. K tomu by měly být definovány dvě metody. Jedna smaže konkrétní soubor, druhá pak všechny soubory s daným tagem.

Jelikož je vytváření souborů z databáze složitá operace, je vhodné jejich počet snížit na minimum. Soubory definující vstupní data úlohy, výstupní data úlohy, vyhodnocovací program, porovnávací program – všechny tyto soubory je zbytečné vytvářet pro každé vyhodnocení znova.

Správným rozšířením funkcionality lze docílit značné úspory počtu vytvářených souborů. Při vytváření souboru je nutno zjistit, zdali se daný soubor má mazat. Toho lze docílit pouze dotazem do databáze. V případě, že se nejedná o soubor se zdrojovým kódem řešení, je vhodné tento soubor ponechat. Jednoduchým nepřiřazením tagu k tomuto souboru lze docílit nesmazání souboru po ukončení vyhodnocení. Je potřeba však mít jedinečný, avšak jednoznačný název souboru pro každý záznam souboru v databázi. Toho je docíleno kombinací ID souboru, jeho přípony a času poslední úpravy záznamu v databázi.

Pro zachování konvence odpovídá přípona souboru příponě originálního souboru. Při nesplnění této vlastnosti se lze potýkat s problémy v případě kompilace souboru překladačem gcc.

6.2.4 Kompilace zdrojových kódů řešení úlohy

Systém vyhodnocovacího serveru by měl být navržen tak, aby v případě potřeby mohl zpracovávat programy napsané v různých programovacích jazycích. Před samotným spuštěním je však vhodné daný program předpřipravit. Jedná se například o kompilaci, jež je nutná pro neinterpretované jazyky.

Kompilace by sice mohla být v samotném sandboxu, je však lepší pro tuto úlohu vytvořit zvláštní modul, jež by jí měl na starost. Před samotnou kompilací je potřeba vybrat správný typ kompileru. Výběr lze rozhodnout podle typu programovacího jazyka, v němž je testovaný program napsán.

Základní verze vyhodnocovacího serveru je tvořena pro práci s programovacími jazyky C a C++. Na unixových systémech je nejběžněji používán kompiler gcc, respektive g++. Kompilace by tedy měla probíhat jako spuštění těchto překladačů a následné kontroly úspěšnosti překladu.

6.2.5 Spuštění testovaného programu v sandboxu

Spuštění testovaného programu za účelem zpracování předem daných vstupních dat je z bezpečnostního hlediska jedna z kritických vlastností celého systému. Sandbox samotný se má starat o zabezpečení spuštění testovaného programu a v případě pokusu o provedení podezřelých, či nebezpečných úkonů program ukončit. Sandbox má dále za úkol hlídat a měřit využité prostředky, tedy paměť, čas a velikost výstupních dat. Ty pak předá vyhodnocovacímu serveru.

Vyhodnocovací server by měl implementovat flexibilní systém spouštění sandboxu. Před každým vyhodnocením řešení úlohy je vybrán správný sandbox na základě typu programovacího jazyka v němž je řešení úlohy implementováno. Jelikož je sandbox implementován jako externí program, je spuštěn spolu se vstupními daty ve formě argumentů.

Ve výstupu sandboxu je očekáván kód vyhodnocení programu, čas běhu programu a maximální velikost využité paměti.

6.2.6 Kontrola správnosti výsledku

Po spuštění testovaného programu se vstupními daty vydá tento program výsledek. Ten je nutno zkontrolovat, zdali je správný.

Nyní vyvstává otázka, jak tuto kontrolu provést. Úloha může mít definovaný správný výsledek jako pomocný soubor. Tato data by se pak mohla kontrolovat pomocí nástroje typu *diff*. Avšak ne všechny úlohy požadují kontrolu výsledku tohoto typu.

Je tedy požadavek na do jisté míry univerzální řešení kontroly výsledku. Toho lze docílit tak, že pro každou úlohu bude existovat zvláštní program, jež porovná správnost výsledku s očekávaným výsledkem. Po získání výstupních dat jsou tato data předána spolu s očekávaným

výsledkem do programu pro vyhodnocení. Ten je zpracuje a vydá výsledek, jež definuje správnost samotného programu.

Soubor s očekávaným výsledkem nemusí obsahovat pouze data výsledku. V podstatě může obsahovat jakákoliv pomocná data, jež jsou potřeba při vyhodnocení správnosti odpovědi. Pokud tento soubor není potřeba, nemusí vůbec existovat. Pro univerzálnost vyhodnocovacích programů je však vhodné ho využít.

6.2.7 Paralelní zpracování úloh

Pro možnost zpracovávání úloh více procesy je nutno při návrhu myslet na některé aspekty, které by mohly v případě opomenutí způsobovat problémy.

Při používání objektů více vlákny je nutno využít synchronizaci. Toto je však úzké hrdlo programu a synchronizace bloků kódu by měla být využita s rozmyslem. Mnohem lepší je vytvořit samostatný objekt pro každé vlákno, nebo navrhnout objekt jako *imutable*[6].

Procesy, jež probíhají mimo modul vyhodnocovacího serveru, je však nutno zabezpečit jiným způsobem, neboť v některých případech se na výše uvedené techniky nelze spolehnout. Jedná se například o komunikaci s databází, či práci se souborovým systémem.

6.2.8 Rozdělení do balíčků

Činnosti jednotlivých úloh je vhodné rozdělit do balíčků, jež budou jednotlivé úlohy implementovat. Balíčky jsou rozděleny převážně podle problémů, které mají řešit. Tyto problémy byly vypsány v kapitole 6.1.

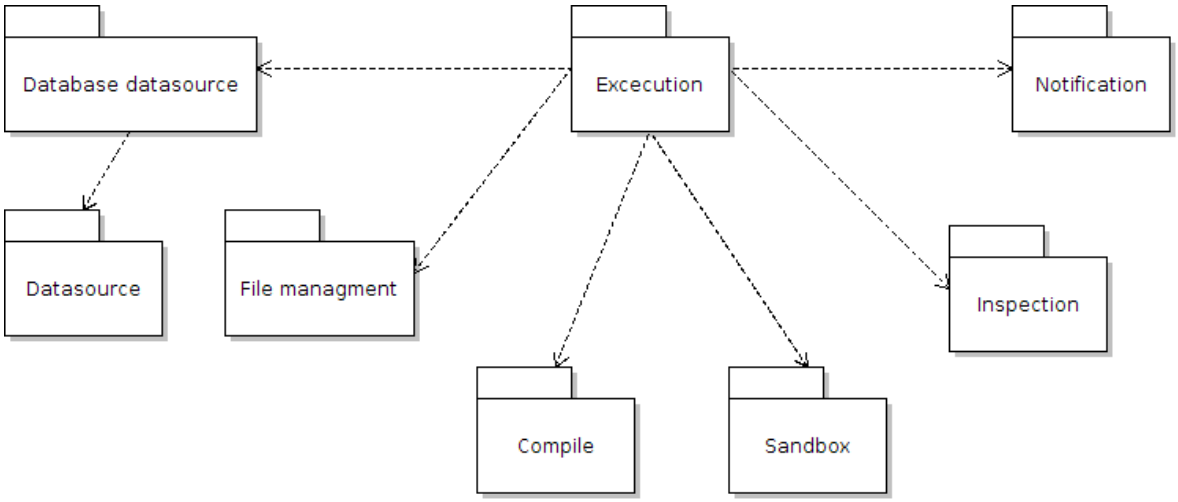
Rozložení a závislosti jednotlivých balíků je vyobrazeno na Obrázku 7. Většina těchto balíčků je autonomních, pouze balíček mající na starost spuštění programu pracuje s ostatními balíčky.

6.3 Použité technologie

Pro implementaci modulu vyhodnocovacího serveru není kladen žádný zvláštní nárok na programovací jazyk, v němž by měl být implementován. Jelikož však byla webová část serveru vyvinuta v programovacím jazyce Java 6.0, byl tento jazyk použit i pro modul vyhodnocovacího serveru.

Pro práci s databází bylo použito standardní rozhraní JPA. Jako implementace pak byl vybrán Eclipselink verze 2.2.0[14]. Konfigurace prostředí byla provedena pomocí Spring frameworku 3.1.1[15].

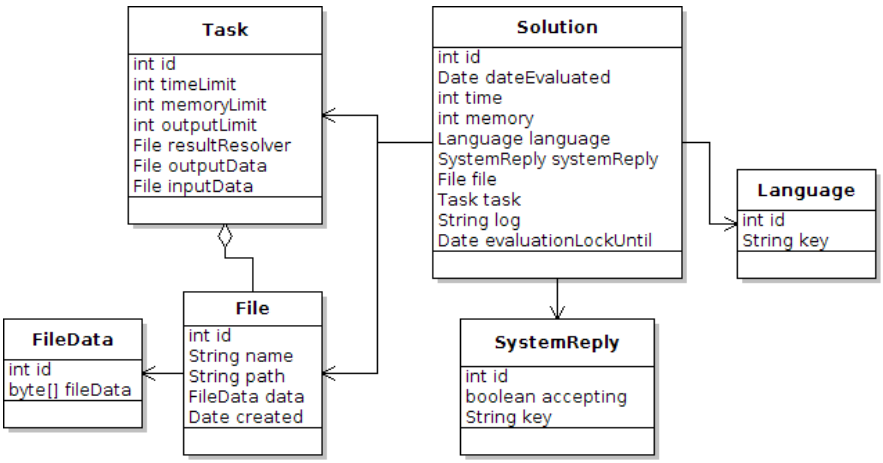
14 Domovskou stránku projektu EclipseLink lze nalézt na adrese http://www.eclipse.org/eclipselink/.
15 Stránku frameworku Spring lze nalézt na adrese http://www.springsource.org/.



Obrázek 7: Rozdělení vyhodnocovacího serveru do balíčků

6.4 Datový model

Pro vyhodnocení daného řešení úlohy je nutno znát nejen zdrojový kód řešení, ale i mnoho dalších informací týkajících se úlohy. Komunikace s datovou vrstvou je do jisté míry izolována rozhraním `DataSource`, je však potřeba ještě navrhnout doménové entity, jež má server obsahovat. Veškeré třídy, které budou tyto entity reprezentovat, pak budou definovány v balíčku `evaluationserver.server.entities` a jejich složení lze nalézt na Obrázku 8.



Obrázek 8: Třídní diagram entit vyhodnocovacího serveru

Nejprve definujeme pomocné entity, tedy například entita `Language`. Ta definuje programovací jazyk, jež může být ve vyhodnocovacím serveru využit. Proměnná `key` zde figuruje jako uživatelský název jazyka, který je používán v konfiguraci.

Entita typu `SystemReply` zastává úlohu výsledku vyhodnocení programu. Proměnná `key` je využita jako název, pomocí něhož se předává typ odpovědi mezi sandboxem a vyhodnocovacím serverem. Proměnná `accepting` indikuje správnost odpovědi.

`File`, neboli soubor uložený v databázi, obsahuje jeho definici. Název `name`, cesta k souboru `path`, datum vytvoření `created`. Navazuje na ní entita `FileData` obsahující samotná data souboru. Tyto entity by sice mohly být uloženy v jedné, avšak data souboru by byla načítána vždy, bez ohledu na jejich použití. Vzhledem k faktu, že soubory mohou mít relativně značnou velikost je tedy entita souboru rozdělena na dvě.

Následuje entita úlohy, tedy `Task`. Ta obsahuje veškeré limity pro řešení dané úlohy. Proměnná `timeLimit` označuje maximální možnou délku zpracování programem, měřeno v milisekundách. Proměnné `memoryLimit` a `outputLimit` označují maximální množství použitelné paměti, respektive maximální délku výstupu, jež program může při řešení vydat, obojí v bytech. Následují soubory `resultResolver`, tedy program pro vyhodnocení správnosti výsledku, `outputData`, pomocný soubor s výstupními daty a `inputData`, tedy vstupní data pro sandbox.

Poslední entita, která je potřebná pro běh vyhodnocovacího serveru, je řešení úlohy, tedy entita `Solution`. Proměnná `dateEvaluated` obsahuje datum a čas startu vyhodnocení úlohy. Proměnné `time` a `memory` obsahují množství času, respektive paměti použité při zpracování vstupu testovaným programem. Proměnné `language`, `systemReply` a `task` jsou odkazy na dříve zmíněné entity k nimž se řešení vztahuje. V proměnné `file` lze nalézt program s řešením a do proměnné `log` se zapisují případné chyby při vyhodnocování úlohy. Konečně proměnná `evaluationLockUntil` je zámek, jež upozorňuje na právě probíhající zpracování úlohy.

6.5 Implementační detaily

V následující části budou rozebrány detaily týkající se implementace modulu vyhodnocovacího serveru. Budou zde také uvedeny detailní řešení problémů, které byly popsány dříve. Veškeré zdrojové kódy se nacházejí v základním balíčku `evaluationserver.server`. Pokud tedy bude odkazováno na nějakou třídu, nebude již tento balíček uváděn. Jednotlivé sekce textu budou postupně následovat běh vyhodnocovacího serveru dle Obrázku 18.

6.5.1 Inicializace

Vstupní bod pro běh serveru je ve třídě `execution.Run`. Ten obsahuje pouze minimální funkcionalitu. Nejprve je vytvořen aplikační kontext, ten se vytváří s pomocí Spring frameworku, respektive Spring DI. Tomu se předá konfigurační soubor *config/configuration.xml*, jež obsahuje kompletní propojení komponent modulu.

Z aplikačního kontextu je získána instance implementace samotného serveru, tedy instance implementující `execution.Server`. Tato instance je spuštěna. Standardní implementace tohoto serveru je třída `execution.ServerImpl`. Ta ve svém konstruktoru přijímá veškeré potřebné závislosti pro běh celého serveru. Tyto závislosti lze nalézt v Tabulce 4.

Typ proměnné	Název proměnné	Popis
DataSourceFactory	dataSourceFactory	zdroj dat
SandboxResolver	sandboxResolver	kontejner se sandbox, získání na základě typu programovacího jazyka
CompilerResolver	compilerResolver	kontejner pro kompilery, získání na základě typu programovacího jazyka
FileManager	fileManager	kompletní správa používaných souborů
Inspector	inspector	třída pro vyhodnocení správnosti výsledku
NotificatorList	notificators	seznam všech notifikátorů
int	threadCount	počet vláken ke spuštění

Tabulka 4: Parametry konstuktoru třídy ServerImpl

V případě předání neplatné hodnoty počtu vláken je zjištěn počet přidělených procesorů JVM a tento počet je nastaven. Po zavolání metody `runServer` je vytvořeno dané množství vláken, jež zpracovávají samotné úlohy. Vlákno třídy `execution.SolutionWorker`, dále jen `SolutionWorker`. Přibližný průběh jeho funkce je vidět na Obrázku 9.

Spuštění běhu vlákna začíná logováním startu. Následuje hlavní cyklus, jež definuje jedno zpracování programu. Do proměnné `tag` je přiřazena unikátní hodnota, které se využívá ke správě souborů[16]. Jelikož je potřeba mít unikátní číslo pro každé vyhodnocení řešení úlohy, je využita třídní proměnná `counter`, jež je inkrementována po každém průběhu vyhodnocení. Tato proměnná je definována jako `volatile`, navíc je při inkrementaci a přiřazení v `synchronized` bloku. Díky tomu je zajištěno požadované chování i při přístupu více vlákny. Nutno podotknout, že by bylo vhodné tuto funkcionalitu vyčlenit do zvláštní třídy. Následuje získání další nevyřešené úlohy.

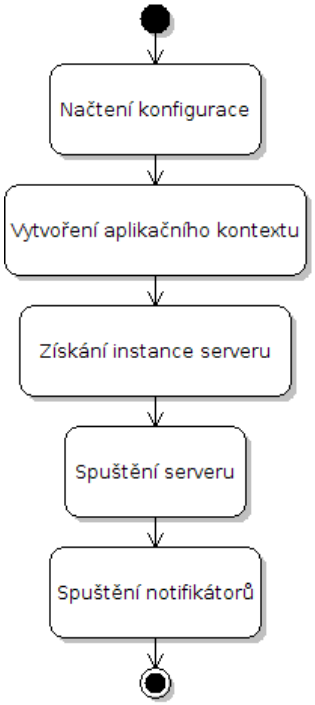
6.5.2 Získání další nevyřešené úlohy

Způsob získání další nevyřešené úlohy je pro vlákno vykonávající vyhodnocení velmi přímočaré. Pro tuto úlohu je implementován speciální objekt implementující rozhraní `java.util.concurrent.BlockingQueue`, dále jen `BlockingQueue`. Toto rozhraní definuje frontu, jež má odlišné chování oproti standardní frontě. Je speciálně navrženo pro úlohy typu producent-spotřebitel[17][2].

Vlákno spotřebitele jednoduše zavolá `solutions.take()`, kde proměnná `solutions` je právě fronta implementující `BlockingQueue`. Návratová hodnota této metody je následující element fronty. V případě nepřítomnosti prvku je aktuální vlákno pozastaveno. V případě výskytu dalšího prvku je vlákno opět probuzeno a pokračuje v běhu.

16 Použití tagů ve správě souborů bylo popsáno v kapitole 6.2.3.

17 Jedná se o problém synchronizace procesů, kde producent produkuje data do zásobníku s omezenou velikostí a konzument je spotřebovává.



Obrázek 9: Aktivitní diagram inicializace vyhodnocovacího serveru

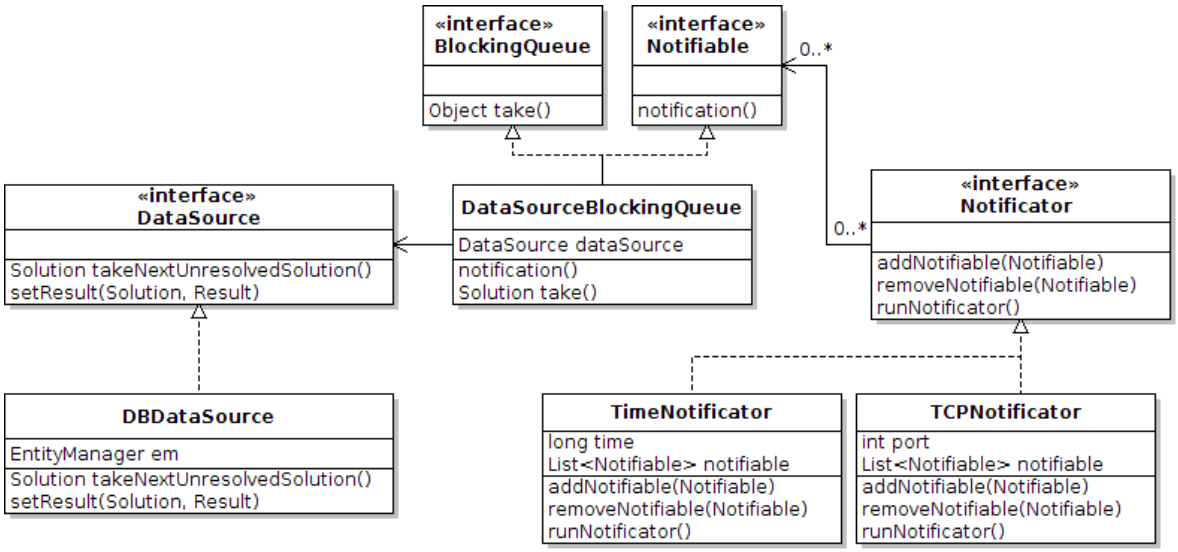
Třída fronty má název `datasource.DataSourceBlockingQueue`. Jedná se o frontu využívající jako zdroj dat objekt implementující `datasource.DataSource`. Fronta navíc implementuje rozhraní `notification.Notifiable`. Toto rozhraní abstrahuje objekt, jež může být informován pomocí notifikátorů[18].

Strukturu tříd spolupracujících s třídou fronty lze nalézt na Obrázku 10.

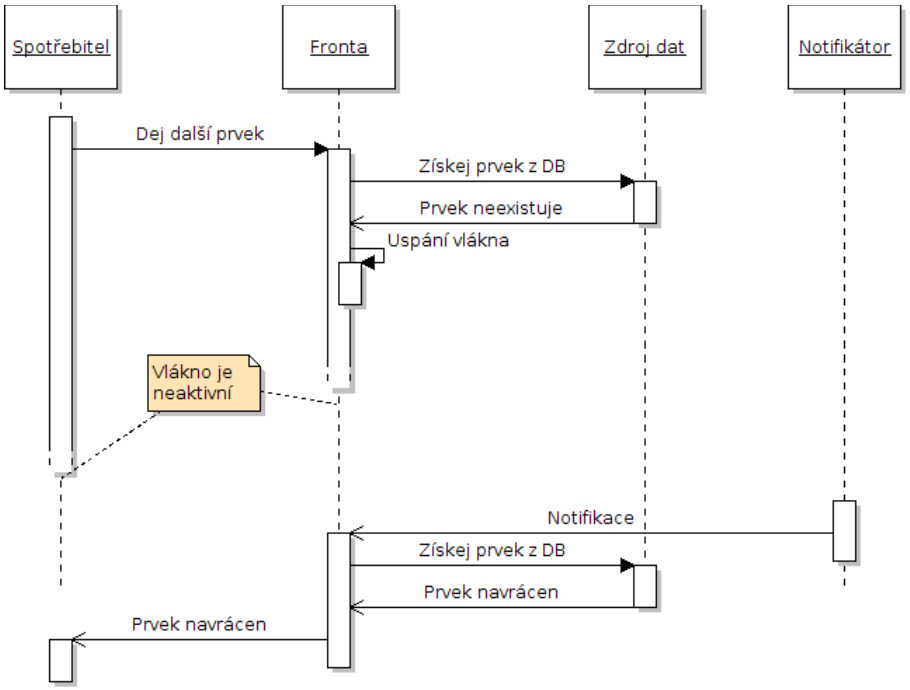
Požadovanou komunikaci lze nalézt na Obrázku 11. Spotřebitel je v našem případě vlákno zpracovávající úlohu. Ten zadá požadavek na získání prvku fronty. Fronta otestuje přítomnost prvku ve zdroji dat, neboli v databázi. V případě neexistence prvku je vlákno uspáno. Probuzení vlákna pokračuje po přijetí notifikace. Fronta opět položí dotaz na existenci prvku ve zdroji dat. V případě existence tento prvek navrátí spotřebiteli.

Jako zdroj dat je zde třída `dbdatasource.DBDataSource`. Ta získává požadovaná data z databáze. Jak již bylo zmíněno dříve, je nutno provést vyhodnocení řešení úlohy atomicky. Toho je docíleno zámkem, respektive atributem `evaluationLockUntil` v entitě definující řešení úlohy. V případě přítomnosti data většího než aktuální v této proměnné je záznam zablokován vůči změnám, respektive vůči vícenásobnému zpracování.

18 Notifikátory byly uvedeny v kapitole 6.2.2.



Obrázek 10: Třídní diagram struktury notifikátorů



Obrázek 11: Sekvenční diagram získání nevyhodnoceného řešení

Pro lepší porozumění bude popsána samotná metoda `takeNextUnresolvedSolution` pro získání nevyřešené úlohy. Nejprve je vytvořen synchronizační blok vůči proměnné entity manageru. Následně je vyslán požadavek do databáze o vyhledání nevyhodnoceného řešení bez zámku. V případě nenalezení takovéhoto záznamu je vrácena hodnota `null`.

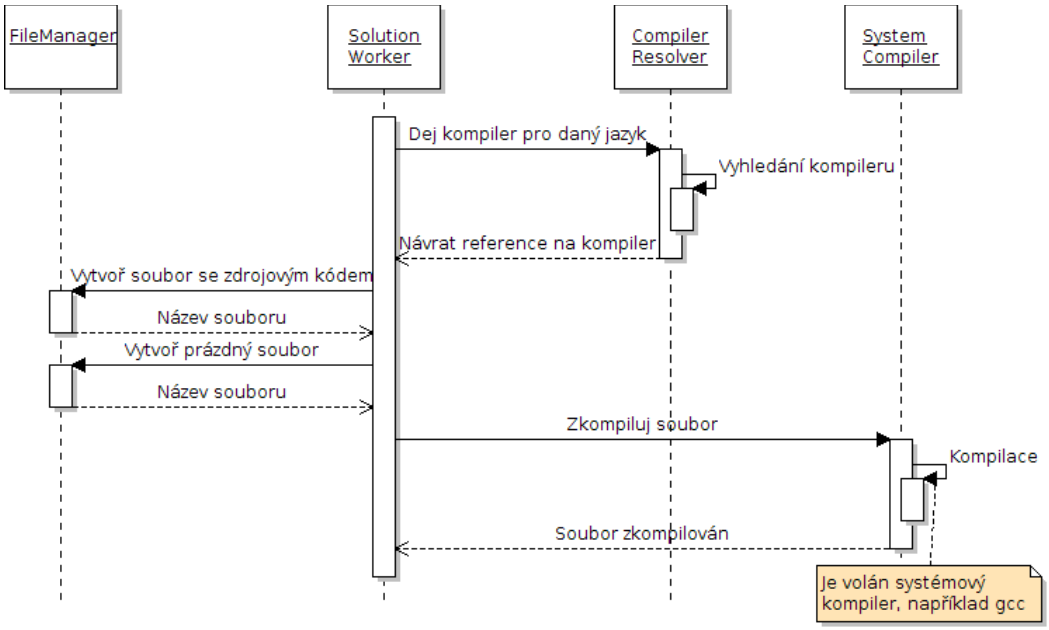
V případě nalezení je započata transakce. Následně je záznam uzamknut v databázi vůči zápisu. Jelikož mezi tímto krátkým intervalem může být záznam upraven jiným vláknem, je provedena kontrola přítomnosti zámku v proměnné `evaluationLockUntil`. V případě její existence je transakce navrácena metodou `rollback` a celá metoda pokračuje od začátku.

Pokud je proměnná `evaluationLockUntil` prázdná, nebo obsahuje datum menší než aktuální, je provedena aktualizace její hodnoty. Entita je synchronizována s databází a transakce je ukončena. Následně je navrácena ke zpracování.

6.5.3 Kompilace programu

Získáním entity řešení úlohy jsou díky závislostem získána veškerá data pro další průběh vyhodnocení. Tento proces následuje kompilace zdrojových kódů řešení úlohy ve spustitelný program.

Ve třídě `SolutionWorker` je zavolána metoda `compile(solution)`, jež má za úkol provést dílčí kroky kompilace. V případě vzniku chyby je tato skutečnost zalogována, entita řešení úlohy je nastavena jako vyhodnocená s kódem *Compile error* a veškeré nepotřebné soubory jsou odstraněny. Následný průběh je zobrazen na Obrázku 12.



Obrázek 12: Sekvenční diagram kompilace zdrojových kódů

Metoda kompilace je zahájena získáním kompileru pro programovací jazyk použitý pro řešení úlohy. Toto má za úkol třída `compile.CompilerResolverImpl`. Zavoláním metody `getCompiler(languageKey)` je získán potřebný kompiler použitý ke kompilaci, v případě že neexistuje je vyvolána výjimka `compile.NoCompilerException`.

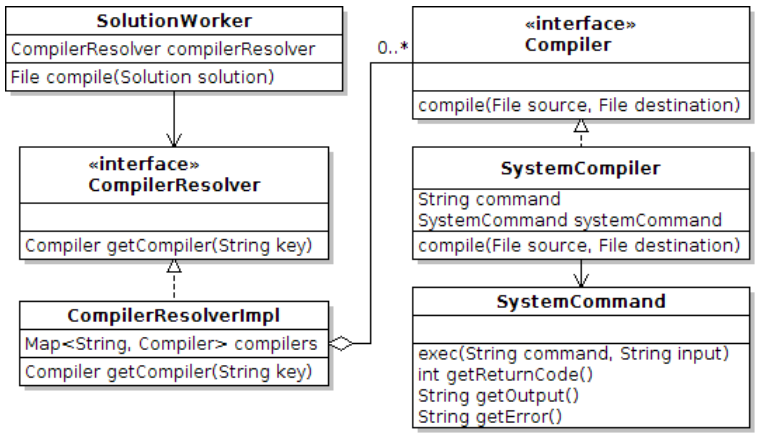
Pomocí proměnné `fileManager` reprezentující správu souborů jsou vytvořeny soubory

se zdrojovým kódem programu a je zarezervován nový soubor pro zkompilovaný program. Následně je využit dříve získaný kompiler k samotné kompilaci programu.

V základu je požadována možnost kompilovat zdrojové kódy programovacích jazyků C a C++. Pro tyto účely je použit například kompiler *gcc*, respektive *g++*. Pro kompilaci je tedy nutno spustit externí program. Pro tento účel je navržena třída `compile.SystemCompiler`.

Při vytvoření instance této třídy je požadován přinejmenším parametr `command`, jež reprezentuje příkaz pro kompilaci zdrojového kódu. Tento příkaz může obsahovat zástupné řetězce `%input%` a `%output%`, jež budou před kompilací nahrazeny za absolutní cestu zdrojového, respektive výstupního souboru. Díky této struktuře je možno v konfiguraci vytvořit libovolné množství kompilátorů využívající tuto třídu.

Pro kompilaci je nutno volat metodu *`compile`*. V případě chyby je vyvolána výjimka `compile.CompilationException`. Pro zjednodušenou práci se spouštěním systémových programů je použita třída `util.SystemCommand`. Ta zapouzdřuje jednoúčelovou činnost skládající se ze spuštění příkazu s daným vstupem, získáním výstupního kódu, výstupu a chybového výstupu. Rozložení tříd vztahujících se ke kompilaci zdrojových kódů lze nalézt na Obrázku 13.



Obrázek 13: Třídní diagram struktury kompilerů

6.5.4 Spuštění programu v sandboxu

Mezi kompilací a spuštěním programu v sandboxu je několik mezikroků, jež je potřeba provést. Nejprve je za pomocí třídy `FileManager` vytvořen, respektive zarezervován, název souboru pro výstupní data programu. Je vytvořen soubor se vstupními daty dané úlohy. Poté je vytvořena proměnná `sandboxSolution`, jež je kontejner se vstupními daty pro sandbox. Název třídy tohoto kontejneru je `Solution`, veškerá data jsou předána v jeho konstruktoru. Vzhledem k většímu počtu parametrů by mohlo být vhodné vytvořit objekt dle návrhového vzoru *builder*[3], jež by jeho vytvoření zprostředkoval.

Následuje volání metody `execute(sandboxSolution)` ve třídě `SolutionWorker`, jež obsahuje zpracování tohoto volání. Jako návratová hodnota je zde instance třídy

`sandbox.ExecutionResult`. Ta obsahuje veškeré potřebné informace týkající se průběhu spuštění programu, tedy odpověď, start, délku trvání, velikost využité paměti, velikost výstupu a log. V případě vzniku chyby je tato skutečnost zalogována, je zanesen daný typ odpovědi k řešení úlohy a jsou odstraněny nepotřebné soubory.

Metoda `execute` má velmi podobnou strukturu dříve popsané metodě `compile`. Díky proměnné `sandboxResolver` je získán daný sandbox dle typu programovacího jazyka. Následně je daný sandbox spuštěn s potřebnými vstupními parametry.

Proměnná `sandboxResolver`, respektive třída `SandboxResolverImpl` je téměř shodná se třídou `CompilerResolverImpl`. Stará se však o třídy implementující rozhraní `Sandbox`.

Implementace sandboxu je ve třídě `SandboxImpl`. Ta v konstruktoru přijímá několik parametrů, důležité jsou však první dva. Jedná se o příkaz, jež má být spuštěn pro spuštění a instance třídy `sandbox.ExecutionResultFactory`, jež převede textový výstup sandboxu do již dříve zmíněné třídy `sandbox.ExecutionResult`. Před spuštěním samotného sandboxu jsou v příkazu nahrazeny zástupné řetězce `%program%`, `%inputData%`, `%solutionData%`, `%timeLimit%`, `%memoryLimit%` a `%outputLimit%`. Toto je provedeno v metodě `prepareCommand`. Následně je s využitím třídy `util.SystemCommand` spuštěn daný příkaz. Ze získaného výstupu je za pomocí `resultFactory` navrácen výsledek spuštění.

Po odeslání návratové hodnoty zpět do třídy `SolutionWorker` je provedena kontrola výstupního kódu, tedy zdali při běhu programu nastala chyba. Pokud nastala, je tato skutečnost zalogována a výsledek je aktualizován. V případě úspěšného spuštění programu následuje vyhodnocení správnosti výsledku.

6.5.5 Vyhodnocení správnosti výsledku

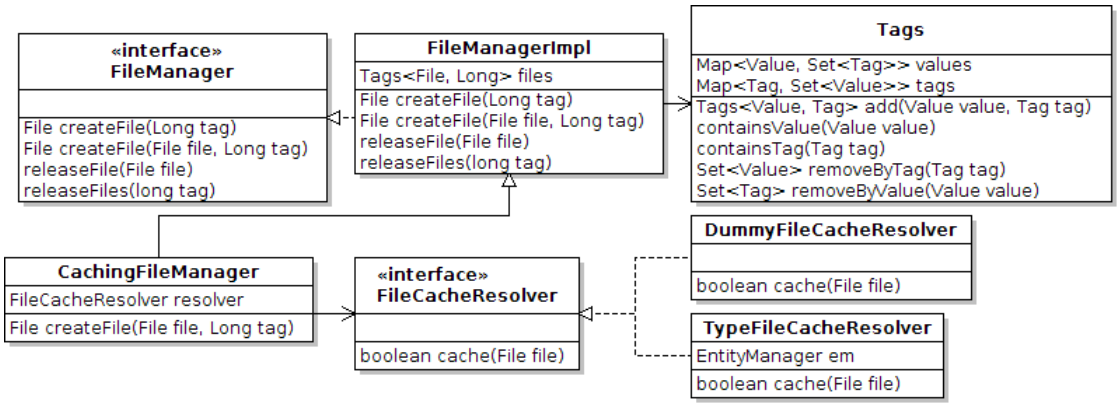
Po zpracování vstupních dat testovaným programem jsou vydána data, jež je nutno zkontrolovat. Jsou vytvořeny další potřebné soubory, tedy výstupní data programu a program pro kontrolu správnosti výsledku. Výstupní data nemusí být přítomna a v tom případě se nevytvářejí. Všechna tato a další potřebná data pro vyhodnocení správnosti výsledku jsou vložena do objektu třídy `inspection.Solution`.

Programu pro kontrolu správnosti výsledku je povoleno spuštění. Následně je zavolána metoda `inspect` pro samotné vyhodnocení výsledku.

V ní je s pomocí třídy `InspectorImpl` spuštěn daný program. Po ukončení běhu programu je ze standardního výstupu programu načten výsledný kód. Ten je finálně zapsán spolu se všemi parametry běhu. Jsou odstraněny všechny nepotřebné soubory, tedy soubory s daným ID tagu. Vyhodnocení tímto končí a vlákno pokračuje získáním další nevyřešené úlohy.

6.5.6 Správa souborů

Třídy, starající se o správu souborů, lze nalézt v podbalíčku `filemanagment`. Závislosti tříd lze nalézt na Obrázku 14. Základní rozhraní má název `FileManager` a definuje veškeré metody práce se soubory, jež byly popsány v návrhu. Jako implementace je zde třída `FileManagerImpl`, jež využívá třídu `util.Tags`. Ta definuje práci s tagy. Ke každé přidané hodnotě lze přidat tag, pomocí kterého lze tuto hodnotu také identifikovat. Každý tag může mít přiřazen více hodnot. V případě odebrání hodnoty je tato hodnota od daného tagu odebrána. Pokud je zavolána metoda pro odebrání dle názvu tagu, jsou odebrány všechny hodnoty, jejichž jediný zbývající tag je odebírán. Díky tomuto systému lze jednoduše pracovat s množinou souborů hromadně, dle požadavků.



Obrázek 14: Třídní diagram správce souborů

Při zavolání metody `createFile` s parametrem tagu je zarezervován nový, nepoužitý soubor v adresáři s dočasnými soubory. Pokud je tato metoda navíc zavolána s entitou definující soubor, je navíc soubor s danými daty okamžitě vytvořen. To značně zjednodušuje práci se soubory, které jsou uloženy v databázi. Zbylé dvě metody odstraní jeden soubor, respektive všechny soubory s daným tagem, jež jsou spravovány.

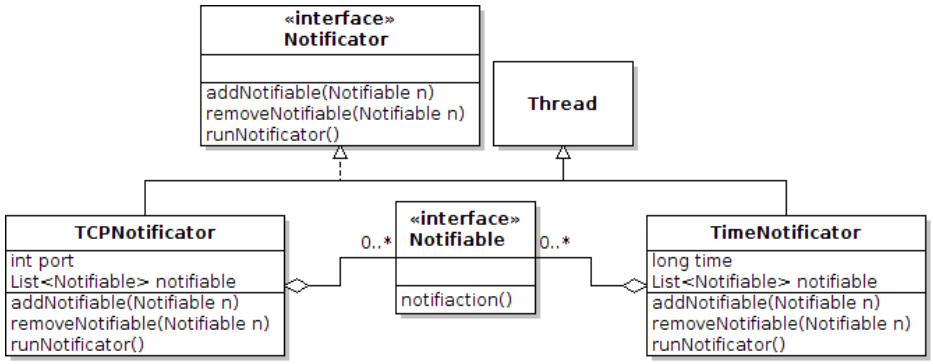
Již dříve byla popsána nutnost uchovávat pracovní soubory na disku ke snížení zátěže serveru. Jedná se o veškeré soubory mimo soubory řešení úlohy. Pro tento účel je zde třída `CachingFileManager`. Ta přepisuje pouze metodu `createFile`. Před vytvořením souboru je nejprve zjištěno, zdali se daný soubor má na konci smazat či ne. Pokud ano, je složen název souboru z jeho ID, časem vytvoření souboru v databázi a příponou. To zajišťuje jeho unikátnost i při případné výměny souboru. Pokud již soubor s tímto názvem existuje, nevytváří se, ale použije se soubor existující. Pokud soubor neexistuje, je s danými daty vytvořen. Důležité je, že soubor není přidán do množiny souborů spravovaných třídou `FileManagerImpl`. Díky tomu se nestane, že by byl tento soubor smazán.

Má-li se soubor na konci zpracování soubor smazat, je použita standardní implementace této metody dle `FileManagerImpl`.

Pro zjištění skutečnosti, zdali se má daný soubor na konci odstranit, je nutno využít externí třídu. Implementace `DummyFileCacheResolver` vrací na tento dotaz vždy kladnou odpověď. Jelikož je však potřeba mazat soubory dle jejich účelu, je nutno vést dotaz na databázi. Je tedy vytvořena další třída v balíčku `dbdatasource` s názvem `TypeFileCacheResolver`, jež vrací kladnou odpověď pouze tehdy, není-li soubor řešením úlohy. To zaručuje požadovanou vlastnost.

6.5.7 Notifikace

Jak již bylo zmíněno v návrhu okamžik, kdy má server zjistit přítomnost nevyhodnoceného řešení, je dán tzv. notifikátory. Byla implementována kombinace časového a síťového notifikátoru, ta jsou znázorněna na Obrázku 15.



Obrázek 15: Třídní diagram notifikátorů

Časový notifikátor po svém spuštění prochází nekonečnou smyčkou. Na začátku každého cyklu je vlákno uspáno na předem danou dobu. Po uplynutí této doby notifikátor předá zprávu zaregistrovaným objektům. Cyklus následně dále pokračuje.

Síťový notifikátor po svém spuštění vytvoří server na předem daném portu. V nekonečné smyčce pak čeká na pokus o navázání komunikace. V případě navázání komunikace je zaregistrovaným objektům předána zpráva a spojení je uzavřeno. Cyklus dále pokračuje.

6.6 Konfigurace

Konfigurace programu je velmi důležitá součást projektu. Ve vyhodnocovacím serveru se konfigurace skládá ze tří souborů, jež jsou umístěny ve složce *config*. Seznam těchto konfiguračních souborů lze nalézt v Tabulce 5.

Soubor	Popis
config.properties	uživatelská konfigurace serveru
logging.properties	nastavení logování
configuration.xml	konfigurační soubor pro Spring DI (IoC)

Tabulka 5: Konfigurační soubory vyhodnocovacího serveru

6.6.1 config.properties

Jedná se o základní konfigurační soubor, jež musí být nastaven. Obsahuje převážně uživatelskou konfiguraci, respektive konfiguraci nutnou pro spuštění a běh serveru. Jedná se například o nastavení připojení k databázi, či cesta ke složce s dočasnými soubory.

6.6.2 logging.properties

Tento konfigurační soubor je standardní soubor využívaný v programovacím jazyce Java, jež obsahuje nastavení o logování. Pomocí hodnot `OFF`, `FINEST`, `FINER`, `FINE`, `CONFIG`, `INFO`, `WARNING`, `SERVE` a `ALL` lze nastavit rozsah logování vzhledem k závažnosti jednotlivých zpráv.

Tyto hodnoty lze nastavit pro každý modul samostatně, jedná se tedy o velmi mocný nástroj při zjišťování chyb, nebo pro záznam jiných informací o běhu. Pro formátování zpráv byla implementována třída `evaluationserver.server.util.ThreadLogFormatter`, jež oproti standardním formátovacím nástrojům přidává do zpráv i název vlákna, které do logu zapisuje.

6.6.3 configuration.xml

Poslední soubor, tedy *configuration.xml,* definuje propojení jednotlivých komponent vyhodnocovacího serveru. Jedná se o konfigurační soubor frameworku Spring. Ten dle návrhového vzoru *Inversion of control* předává závislosti objektů v konstruktoru, nebo pomocí setterů[10]. Pomocí tohoto souboru lze nastavovat pokročilé vlastnosti serveru, jež nejsou v souboru *config.properties* dostupné.

6.7 Testování

Pro modul vyhodnocovacího serveru byly napsány testy ověřující předpokládanou funkčnost jednotlivých tříd. Jedná se o jednotkové testy.

Při psaní testů byly použity nástroje, jež značně zjednodušují a zpřehledňují jejich psaní. Jedná se o testovací framework JUnit 4.8.2[19] a mockovací knihovnu JMock[20].

Pro spuštění testů je možno využít vývojového prostředí, které přehledně vypíše výsledky testů. Je také možno ke spuštění testů využít příkazovou řádku. Ve složce modulu vyhodnocovacího serveru je nutno spustit následující příkaz:

```
ant test
```

Po jeho spuštění je projekt zkompilován a na konzoli jsou zobrazovány výsledky testů.

6.8 Programy pro kontrolu správnosti výsledku

Pro kontrolu správnosti výsledku je nutno vytvořit program, který vyhodnotí vstupní data,

19 Domovskou stránku projektu JUnit lze nalézt na adrese http://www.junit.org/.
20 Domovskou stránku knihovny JMock lze nalézt na adrese http://www.jmock.org/.

výstupní data a testovaná data. Po jejich načtení a zpracování musí na výstup vydat výsledek. Základní kostru takovéhoto programu lze nalézt v souboru *tracer/diff/skeleton.c*. Je naprogramována v jazyce C a jedná se o jednosouborový program.

V hlavní funkci tohoto programu je nejprve provedena kontrola vstupních parametrů. Ty jsou definovány jako názvy souborů výstupu testovaného programu, vstupu a výstupu úlohy. Výstup není u úlohy povinný a v tomto případě je jeho hodnota nastavena na `NULL`. Následně jsou tyto soubory otevřeny, zkontrolovány chyby a je zavolána funkce pro zpracování výsledku. Ta má název `process` a měla by vrátit jednu z předdefinovaných konstant `ACCEPTED`, `PRESENTATION_ERROR`, `WRONG_ANSWER`, nebo `INTERNAL_ERROR`.

Po navrácení jedné z těchto hodnot jsou uzavřeny všechny otevřené soubory a na standardní výstup je vypsána zkratka těchto konstant, jež dále zpracuje samotný vyhodnocovací server.

7 Sandbox

7.1 Úlohy modulu

Při výběru funkcí, jež má tento modul obsahovat, byl jeden hlavní požadavek, a to zajistit co největší bezpečnost během spouštění soutěžního programu. Tato vlastnost je klíčová pro spolehlivý běh vyhodnocovacího serveru.

Soutěžící se mohou v průběhu soutěže snažit všemi možnými způsoby systém obelstít, aby vyhodnotil jejich odeslaný program jako správný. Mohou se pokoušet získat soubory ze serveru, mohou se též pokusit navázat síťové spojení. Nebo spustit program vícevláknově pro zrychlení běhu jejich programu. Všechny tyto potencionální útoky je potřeba identifikovat a navrhnout způsob, jak je detekovat a zamezit v jejich provedení.

Jako základní vstupní parametry pro tento modul jsou:

- název souboru testovaného programu
- název souboru se vstupními testovacími daty
- název souboru pro zápis výstupních testovacích dat
- časový limit pro výpočet
- paměťový limit pro výpočet
- limit výstupu

Dále jsou tomuto modulu předány volitelné parametry jako název logovacího souboru a další.

Po vyhodnocení výsledku, ať už testovaný program vydal správný výstup či ne, musí být předány zpět informace o výsledném běhu programu. Tyto informace zahrnují:

- informace o chybách během provádění programu
- výstup programu
- čas využitý programem
- maximální množství použité paměti

Výstup může zahrnovat také další užitečné informace o průběhu vykonávání programu například logovaná data, apod. Položky jako čas provádění programu, či velikost využité paměti nejsou nezbytně nutné pro ostatní moduly, avšak poskytují detailnější informace o kvalitě testovaného programu.

7.2 Návrh modulu

Jak již bylo zmíněno dříve, základní úloha tohoto modulu je izolace spouštěného programu od okolí a zabezpečení tak systému proti případným útokům. Jelikož požadavky funkcionality vyhodnocovacího serveru zahrnují také zamezit použití nepovolených funkcí, byla tato

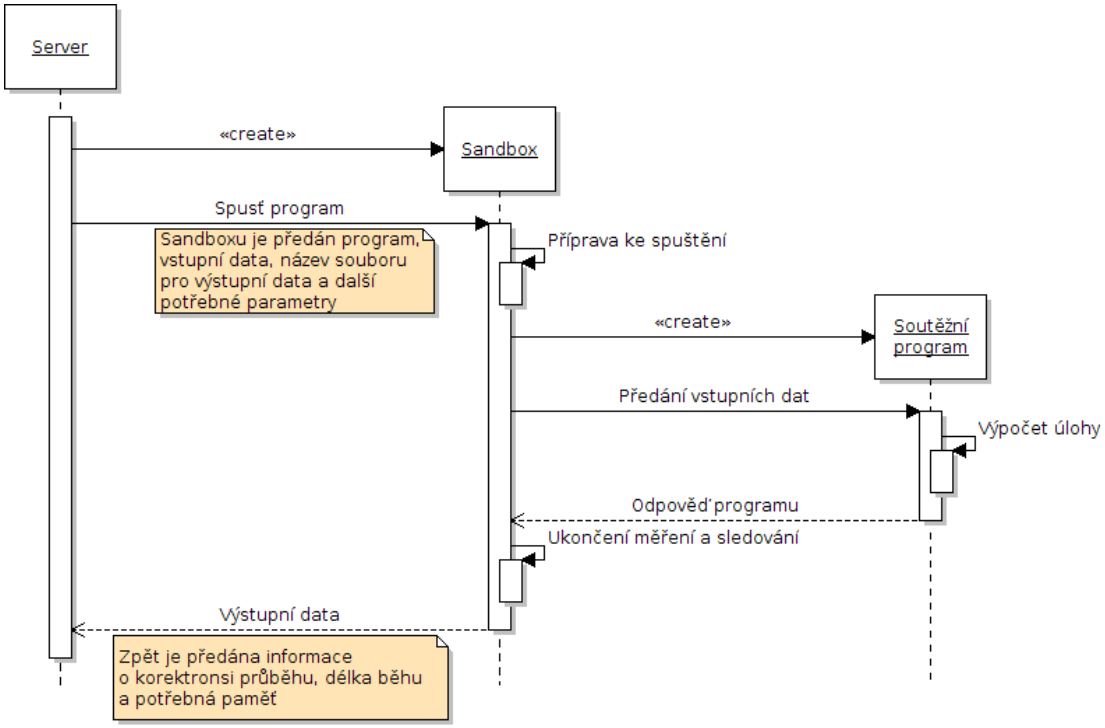
funkcionalita přidělena právě modulu sandbox.

Ve vyhodnocovacím serveru je možno použít programovací jazyk C nebo C++. Tento modul se tedy bude věnovat těmto dvěma jazykům. V případě potřeby využití jiného programovacího jazyka je nutné upravit snadbox pro daný programovací jazyk a zajistit kompatibilitu s ostatními moduly.

Sandbox musí zamezit provedení nepovolených úkonů. Jelikož by sestavení tohoto seznamu bylo značně obtížné a bylo by téměř jisté, že by tento seznam neobsahoval vše, je lepší sestavit seznam úkonů, které daný program provádět může. Výčet těchto úkonů je následující:

- číst ze standardního vstupu
- zapisovat na standardní výstup, s daným omezením délky výstupu
- alokovat si staticky či dynamicky omezené množství paměti
- využívat standardních matematických funkcí daného jazyka
- využívat základních konstrukcí daného jazyka
- program smí běžet pouze v daném časovém limitu

Standardní průběh spuštění daného programu sandboxem je znázorněn na Obrázku 16.



Obrázek 16: Sekvenční diagram spuštění testovaného programu

Z důvodu nutnosti kontroly veškerých činností soutěžního programu je vhodné použít pomůcku pro sledování systémových volání operačního systému. Díky tomu je možno kontrolovat běh klientského programu a reagovat na nepovolené činnosti, jež by program mohl provést.

7.2.1 Vstupní a výstupní data

Soutěžní program čeká vstupní data na standardním vstupu a výstupní data zapisuje na standardní výstup. Je tedy nutno zdroj dat přesměrovat, aby je sandbox mohl ovládat. Před samotným spuštěním se tedy přesměruje do standardního vstupu soubor, který je předán sandboxu jako parametr. Taktéž standardní výstup je přesměrován do souboru.

V potaz je také nutno brát omezení množství dat zapsaných na standardní výstup. Lze jej řešit například pomocí průběžné kontroly velikosti výstupního souboru. Operační systém však nabízí elegantnější řešení a to nastavení maximální velikosti souboru, jež daný proces může vytvořit.

Docílí se ho pomocí metody `setrlimit` s prvním parametrem `RLIMIT_FSIZE`. Pokud by se klientský program pokusil zapsat na výstup větší množství dat než je povolený limit, byl by program ukončen se signálem `SIGXFSZ`, tedy s překročením maximální velikosti souboru. Tento signál lze pak zachytit v sandboxu a ten vrátí informaci o této skutečnosti vyhodnocovacímu serveru.

Vzhledem k tomu, že je preferován výčet systémových volání, jež mají být povoleny a ostatní jsou zakázány, musíme přidat výjimku pro čtení a zápis ze standardního vstupu a na standardní výstup. V případě zachycení systémového volání `SYS_read` je nutno otestovat, zdali je požadováno čtení ze standardního vstupu. Pokud ano, je nutno toto čtení povolit. Obdobně toto platí i pro systémové volání `SYS_write` a standardní výstup.

7.2.2 Využití paměti

Při běhu programu je nutno sledovat množství využité paměti, aby jí testovaný program nezabral více, než je povoleno. Tuto kontrolu je možno provést také pomocí metody `setrlimit` s parametrem `RLIMIT_AS`. Ta by v případě překročení paměti povolené limitem ukončila program.

Tento postup však neřeší zjištění maximální velikosti paměti využité programem. Pro jeho zjištění je nutno kontrolovat jeho aktuální využití paměti a ukládat jeho maximální stav. V případě překročení maximálního limitu pak program ukončit.

Velikost aktuální použité paměti je zjištěna ze souboru */proc/PID/stat*. Z tohoto souboru lze vyčíst mnoho užitečných informací o daném procesu, pro jednoduchost však sandbox zjišťuje pouze velikost využité paměti daným procesem.

7.2.3 Soubory

Přístup k jakýmkoliv externím souborům je naprosto nepřijatelný. Je tedy nutno pokus o otevření souboru zastavit před jeho provedením. Pokus o otevření souboru lze zachytit pomocí systémového volání `SYS_open` a `SYS_openat`.Bohužel, tato systémová volání zachytávají i pokusy o přístup k externím knihovnám. Zde je nutno rozlišovat, zdali se uživatel snaží využít povolené, standardní knihovny, či nepovolené, k níž má být přístup zamítnut.

Je tedy nutno definovat výčet knihoven, jež mohou být programem používány. Standardně povolené knihovny lze nalézt v Tabulce 6.

Soubor	Funkce
ld.so.cache	cache načítaných knihoven
libc.so.6	standardní knihovna jazyka C
libstdc++.so.6	standardní knihovna jazyka C++
libm.so.6	matematická knihovna
libgcc_s.so.1	knihovna využívaná při překladu pomocí překladače gcc

Tabulka 6: Knihovny povolené sandboxem

Jakýkoliv pokus o otevření těchto souborů musí být povolen, k ostatním musí být přístup odepřen. Dále je nutno povolit načtení hlaviček těchto knihoven v systémovém volání `SYS_read`.

Seznam těchto knihoven není finální a obecně mohou být v některých případech tyto soubory jinak pojmenovány. Je tedy potřeba mít možnost specifikovat tyto soubory externě.

7.2.4 Časový limit

Po spuštění testovaného programu je nutno kontrolovat také čas, po který běží. V případě překročení je potřeba běžící proces ukončit a informovat o této skutečnosti vyhodnocovací server.

Je pravda, že by měření mohl provádět modul vyhodnocovacího serveru. Sandbox však před samotným spuštěním testovaného programu provádí inicializaci měření a další úkony, které by do měřeného časového úseku neměly být započteny. Navíc by vyhodnocovací server měřil absolutní čas od spuštění sandboxu po jeho ukončení. V případě různě zatíženého serveru by tedy časy testovaného programu nutné pro výpočet výsledku byly značně odlišné.

Systém však nabízí mnohem zajímavější řešení. Lze měřit virtuální čas, tedy čas, jež procesor skutečně strávil při provádění procesu. Odpadne tedy nepříjemná situace, ve které by stejný testovací program jednou prošel a podruhé ne.

Pro toto řešení je použita funkce `setitimer` s parametrem `ITIMER_VIRTUAL` a časovým intervalem, jež má být přidělen programu. V případě překročení intervalu je vyslán signál `SIGVTALRM`, který sandbox zachytí, proces ukončí a odešle odpověď vyhodnocovacímu serveru.

S tímto řešením je však spojen jeden nepříjemný problém. V případě, že by testovaný program neustále čekal na vstupní data, program by nikdy neskončil. Do takto měřeného času se totiž čekání na vstup nezapočítává. Je tedy nutno přidat kontrolu reálného času, kdy program běží. Tento čas by měl být nějakým způsobem navýšen, aby zde figuroval pouze jako sekundární kontrola proti deadlocku.

Přestože použití měření virtuálního času zajišťuje lepších výsledků, délku tohoto času se mi nepodařilo zjistit. Do vyhodnocovacího serveru se tedy posílá reálný čas od spuštění do ukončení programu. Mohou tedy existovat případy, kdy navenek program překročil časový limit, avšak byl

přijat. Férové vyhodnocení se mi však zdá důležitější, než případné nejasnosti ve výsledcích.

7.3 Použité technologie

Při návrhu sanboxu pro programovací jazyk C a C++ bylo využito značné množství funkcí spojených s operačním systémem. Jedná se o dosti nízkoúrovňové funkce a použití programovacího jazyka Java by bylo značně nepraktické a to i z hlediska výkonu.

Po zvážení všech známých faktů jsem se rozhodl pro tento modul použít programovací jazyk C. Snadnější použití prostředků operačního systému je však na úkor nutnosti použít dva programovací jazyky pro implementaci serveru.

Jako základní prvek modulu sandbox je vhodné uvést funkci `ptrace`, jež zajišťuje odchytávání systémových volání testovaného programu. Tato funkce je zřejmě nejběžněji používaná v debuggerech, kde zajišťuje odchytávání činnosti klientského programu.

7.4 Implementační detaily

V následující části bude podrobněji rozebrána samotná implementace modulu sandboxu pro programovací jazyky C a C++. Pro lepší přehled je na Obrázku 19 zobrazen zjednodušeně průběh tohoto modulu.

7.4.1 Rozdělení do souborů

Samotný modul sandboxu je rozdělen celkově do 11 souborů, z toho 6 souborů je hlavičkových. Seznam zdrojových souborů jazyka C, v němž je tento modul implementován je v Tabulce 7.

Soubor	**Funkce**
call_checker.c	kontrola jednotlivých systémových volání
debug.c	logování činností během kontroly testovaného programu
limits.c	kontrola překročení limitů stanovených pro řešení
main.c	vstupní bod a inicializace programu
proc_stat.c	pomocný soubor jež obsahuje funkci pro zjištění aktuální velikost využité paměti daným procesem z */proc/PID/stat*

Tabulka 7: Zdrojové soubory modulu sandbox

Ke každému uvedenému zdrojovému souboru existuje také hlavičkový soubor, v němž jsou definovány funkce použitelné ostatními soubory, případně také struktury a konstanty používané v ostatních souborech. Mimo tyto hlavičkové soubory je zde ještě jeden s názvem *responses.h*, jež obsahuje sadu konstant odpovídajících výstupním stavům sandboxu, tedy například `ACCEPTED`, či `MEMORY_LIMIT_EXCEEDED`.

7.4.2 Start sandboxu

Běh sandboxu začíná funkcí `main` v souboru *main.c*. Této funkci jsou předány argumenty s nimiž sandbox dále pracuje. Jedná se konkrétně o tyto parametry:

- název testovaného programu
- název souboru se vstupními parametry, tento soubor je přesměrován do standardního vstupu testovaného programu
- název souboru pro zápis výstupu, do tohoto souboru je přesměrován standardní výstup testovaného programu
- časový limit pro běh programu v milisekundách
- paměťový limit testovaného programu v bytech
- maximální délka výstupu testovaného programu v bytech
- název souboru pro zápis logu, v případě uvedení znaku mínus bude log zapisován do chybového výstupu sandboxu
- název souboru se seznamem povolených knihoven, jež může testovaný program využívat, v případě uvedení znaku mínus bude použit seznam uvedený v Tabulce 6.

Program začíná kontrolou počtu vstupních parametrů a dále je vytvořena proměnná `limit` typu `Limits`. Ta je naplněna hodnotami vstupních limitů. Následně je proveden příkaz `fork`, tedy kopie aktuálního procesu, jež rozdělí běh programu na samotný sandbox a testovaný program. V případě selhání systémového volání *fork* je vypsán chybový stav a proces sandboxu je ukončen s návratovým kódem 1 značícím chybu.

7.4.3 Proces testovaného programu po systémovém volání fork

Po úspěšném dokončení systémového volání *fork* má proces potomka za účel spuštění testovaného programu.

Proces tedy pokračuje inicializací proměnné `limit` a následné inicializaci funkcí provádějících kontrolu těchto limitů. Následně se přesměruje soubor daný argumentem do standardního vstupu testovaného programu a jeho standardní výstup se přesměruje do souboru specifikovaného taktéž v argumentu.

V případě jakékoliv chyby při výše zmíněných úkonech se provede vypsání chyby na chybový výstup a program se ukončí s návratovým kódem 1. V případě, že vše proběhlo v pořádku, je spuštěn testovaný program. Veškerý průběh tohoto procesu je nyní v režii testovaného programu.

Předpokládané chování testovaného programu je takové, že si ze standardního vstupu načte data, zpracuje je a výsledek vypíše na standardní výstup. Tyto úkony nemusejí být striktně v tomto pořadí. S ukončením programu by měl být vydán návratový kód 0, indikující úspěšné ukončení běhu programu.

7.4.4 Proces sandboxu po systémovém volání fork

Po úspěšném dokončení systémového volání *fork* má proces rodiče za úkol kontrolovat proces testovaného programu.

Proces zkontroluje, zdali je požadováno načtení seznamu povolených knihoven z externího souboru. Pokud ano, je zavolána funkce `load_libraries` ze souboru *call_checker.c*. Ta má za úkol tento seznam načíst. Pro jednoduchost je seznam knihoven definován jako statické pole délky 20 s maximální délkou řetězce 100. V případě nutnosti je potřeba změnit hodnotu těchto konstant. V případě jakékoliv chyby je vypsána do standardního výstupu a program je ukončen.

Následně je inicializován nástroj pro logování běhu testovaného programu. V případě zadání logovacího souboru je veškeré logování směřováno do tohoto souboru. V případě, že žádný soubor uveden není, je logováno do chybového výstupu. Každý záznam logu má následující tvar:

`datum čas – název – prefix – text`

Jako název je vyplněn název logované události a místo prefixu je doplněn název testovaného programu. Poté je inicializována proměnná `limit`, do které je přiřazeno id procesu testovaného programu.

Následuje volání funkce `trace` starající se o samotné sledování testovaného programu.

7.4.5 Sledování testovaného programu

Sledování začíná inicializací pomocných proměnných a dále nekonečnou smyčkou. První příkaz v této smyčce `wait(&status);` čeká na systémové volání testovaného programu. Do proměnné `status` je přiřazena hodnota, díky níž je možno zjistit detailnější informace o příčině ukončení čekání.

Následuje zjištění, zdali je testovaný program ukončen. Pokud ano, je na standardní výstup vypsána informace o úspěšném běhu testovaného programu spolu s naměřenými hodnotami. Pokud testovaný program ukončen není, je testována hodnota proměnné `status`, zdali je přítomen signál ke zpracování. Následně proběhne akce dle typu signálu.

Jedná-li se o signál typu `SIGXFSZ`, byl překročen limit maximální velikosti souboru[18]. Jinak řečeno, program vypsal příliš mnoho dat do standardního výstupu. Proces testovaného programu je ukončen a na výstup je vypsána informace o překročení výstupního limitu.

Jedná-li se o signál typu `SIGVTALRM`, `SIGALRM`, nebo `SIGPROF`, byl překročen časový limit pro běh testovaného programu[17][13][14]. Proces testovaného programu je ukončen a na výstup je vypsána informace o překročení časového limitu.

V případě signálu typu `SIGSEGV` se jedná o neoprávněný přístup do paměti, což je bráno jako běhová chyba testovaného programu[15]. Proces testovaného programu je ukončen a tato informace je vypsána na výstup.

K testování aktuálně použité paměti nelze využít žádný speciální signál. Proto je využit

signál `SIGTRAP`, který je volán v případě vložení breakpointu, tedy v případě výskytu systémového volání[16]. Při tomto signálu je zjištěn aktuální množství použité paměti. Při překročení maximální povolené velikosti je testovaný program ukončen a informace je vypsána na standardní výstup.

Následuje kontrola samotného systémového volání. Pokud tato kontrola vyhodnotí systémové volání jako zakázané, je testovaný program ukončen a na výstup se vypíše informace o pokusu provedení nepovoleného systémového volání.

7.4.6 Kontrola systémového volání

Nejprve se zjistí typ systémového volání pomocí funkce `ptrace` s parametrem `PTRACE_PEEKUSER`. Dále je nutno zjistit aktuální stav systémových registrů, toho je dosaženo pomocí volání funkce `ptrace` s parametrem `PTRACE_GETREGS`. Následuje samotná kontrola systémového volání ve funkci `check_call` v souboru *call_checker.c*. Ta zkontroluje typ systémového volání pomocí registru `eax`.

Jedná-li se o systémové volání typu `SYS_read`, je zkontrolován soubor, z něhož je čteno. Jedná-li se o standardní vstup, nebo jsou-li čteny hlavičky potřebných knihoven, je volání vyhodnoceno jako povolené. V ostatních případech je volání vyhodnoceno jako nepovolené a do logu je zapsána chyba. Následně je funkce ukončena s návratovým kódem `RESTRICTED_FUNCTION`.

V případě volání `SYS_write` a `SYS_writev` je kontrolován pouze registr `ebx`, tedy soubor, do něhož se zapisuje[26]. Pokud se jedná o standardní výstup, je vše v pořádku. V ostatních případech je funkce ukončena s kódem `RESTRICTED_FUNCTION`. Dále jsou bez další kontroly povolena systémová volání uvedená v Tabulce 8.

Název	**Funkce**
SYS_close	uzavření souboru
SYS_access	kontrola přístupových práv k souboru
SYS_brk	alokace paměti
SYS_mmap	alokace paměti
SYS_munmap	dealokace paměti
SYS_mprotect	nastavení ochrany stránek paměti
SYS_mmap2	alokace paměti
SYS_set_thread_area	nastavení úložiště pro aktuální vlákno
SYS_exit_group	ukončení všech vláken procesu
SYS_fstat64	zjištění informací o souboru
SYS_stat64	zjištění informací o souboru

Tabulka 8: Další povolená systémová volání pro architekturu x86

Na 64-bitových operačních systémech je nutno povolit ještě systémové volání `SYS_arch_prctl` a změnit, či odstranit několik dalších.

Poslední dva povolené typy systémového volání jsou `SYS_open` a `SYS_openat`. Kontrola těchto systémových volání je složitější, jelikož je potřeba kontrolovat i soubor, jež je otevírán. Nejprve je získán název souboru pomocí volání funkce `get_ptrace_text`. Ta pomocí volání funkce `ptrace` s parametrem `PTRACE_PEEKDATA` postupně získá celý název souboru[9] [12]. Následně se ve funkci `check_library` zjistí, zdali je tento soubor povolen k otevření. Nutno podotknout, že kontrola názvu souboru je provedena bez cesty, respektive se kontroluje, jestli nějaká povolená knihovna je suffixem otevíraného souboru.

7.5 Testování

Jelikož je modul sandboxu kritický pro bezpečnost serveru, jsou vytvořeny testy, které mají kontrolovat jeho správnou funkci v daných případech. Každý z nich je definovaný jako soubor se zdrojovým kódem a kódem, jež má být definován jako správná odpověď sandboxu. Seznam těchto testů je vypsán v Tabulce 9. Tyto testy jsou umístěny ve složce *tracer/tests/tests/*.

Název	**Popis**	**Očekávaný výsledek**
fopen	pokus o otevření souboru fopen.cpp	restricted function
fork	pokus o provedení forku	restricted function
memory	alokace nadměrné množství paměti	memory limit exceeded
memory_malloc	alokace nadměrné množství paměti pomocí funkce malloc	memory limit exceeded
output	výpis velkého množství textu na výstup	output limit exceeded
printf	standardní výpis krátkého textu	accepted
scanf	standardní načtení čísel ze vstupu	accepted
sigsegv	přístup do nepovolené části paměti	runtime error
time	překročení časového limitu pomocí cyklů	time limit exceeded
time2	čekání na vstup	time limit exceeded

Tabulka 9: Popis testů modulu sandboxu

Pro testy je použit seznam knihoven definovaný v Tabulce 6. V případě potřeby je možno tento seznam změnit v souboru *tracer/tests/libraries*.

Jako vstup je použit seznam čísel od 1 do 5, každé na samostatném řádku. Pro provedení všech testů je potřeba spustit soubor *tracer/tests/runtests.sh*. Tento skript postupně provede všechny výše uvedené testy a vypíše správnost jejich provedení. Pro správnou funkci je potřeba nejdříve zkompilovat samotný tracer v *RELEASE* módu. Toho lze docílit například spuštěním skriptu *build.sh* v kořenové složce projektu. Ten mimo jiné připraví veškeré potřebné moduly pro běh serveru[21].

21 O sestavení systému pojednává kapitola 8.2.

Po spuštění testů je na obrazovku vypsán výstup indikující správný průběh testů. V případě úspěchu všech testů je počátek výstupu následující:

```
Compiling ...
Processing 10 tests
Test [printf] with expected result AC
Code: AC Time: 6ms Memory: 3055616b - OK

Test [scanf] with expected result AC
Code: AC Time: 14ms Memory: 3055616b - OK

Test [fopen] with expected result RF
02.04.2012 11:53:04 - Error - fopen - RESTRICTED FILE OPEN
02.04.2012 11:53:04 - Error - fopen -  --> Argument: 'fopen.cpp'
Code: RF Time: 5ms Memory: 3117056b - OK
```

V případě opakovaného spuštění se mohou hodnoty délky běhu testů, využití paměti a data lišit.

7.6 Možná rozšíření snadboxu do budoucna

Modul sandboxu v aktuálním stavu je plně dostačující pro spouštění testovaných programů implementovaných v jazyce C a C++. Pro zvýšení bezpečnosti je však možno využít ještě dalších funkcionalit operačního systému.

7.6.1 Chroot

Jedna z možností posílení bezpečnosti sandboxu je využití operace *chroot*. Tato operace vytvoří nový virtuální kořen souborového systému, do něhož má daný proces a jeho potomci přístup. V případě prolomení bezpečnostních ochran sandboxu se tedy dostane útočník pouze do virtuálního systému souborů, jež nenabízí takové možnosti jako plnohodnotný systém souborů. V tomto virtuálním adresáři by se nacházelo pouze nezbytně nutné množství dodatečných souborů, jež by proces pro svůj běh vyžadoval. Případný úspěšný útok by tedy způsobil pouze minimální škody, jelikož útočník by neměl možnost přistupovat k souborům mimo definované.

Chroot však není původně určen jako bezpečnostní prvek, avšak ve spojení spolu se sandboxem by tvořil robustní systém zabezpečení.

7.6.2 Sandbox pro jiné kompilované jazyky

Implementovaný sandbox je testován a určen pro jazyky C a C++. V případě potřeby využívat jiné kompilované jazyky by bylo nutno sandbox lehce upravit. Jednalo by se však převážně o úpravu souboru s povolenými knihovnami, jelikož nepovolená systémová volání sandbox kontroluje.

7.6.3 Sandbox pro interpretované jazyky a jazyky využívající běhové prostředí

V případě potřeby použití jazyka interpretovaného, jako je například PHP, nebo jazyka využívajícího běhové prostředí, například Java, by bylo nutno zvolit značně odlišný způsob implementace modulu sandbox.

Bylo by nutno rozlišovat mezi operacemi jež vyžaduje samotný interpreter, či běhové prostředí a mezi operacemi jež provádí samotný testovaný program. Jako řešení se zde nabízí upravit knihovny těchto jazyků tak, aby obsahovaly pouze povolený seznam příkazů, které testovaný program může využít.

K monitorování využití systémových prostředků by šly využít například informace získané z běhového prostředí.

8 Instalace

Následující text detailně popíše postup instalace systému vyhodnocovacího serveru. Níže uvedený postup byl vyzkoušen na nově nainstalovaném operačním systému Debian 6.0.4 i386, avšak s drobnými úpravami by měl být platný i pro jiné operační systému GNU/Linux. Veškeré příkazy je potřeba provést v příkazovém řádku.

8.1 Zdrojové kódy vyhodnocovacího serveru

Pro kompilací a spuštěním vyhodnocovacího serveru je nejprve nutno získat jeho zdrojové kódy.

8.1.1 Zkopírování zdrojových kódů z přiloženého CD

Na přiloženém CD jsou uloženy zdrojové kódy vyhodnocovacího serveru. Lze je nalézt ve složce *source*.

8.1.2 Stažení z webu

Pro získání nejnovější verze vyhodnocovacího serveru je upřednostňováno získání verze z webu. Adresa na níž je server umístěn je *https://github.com/Foowie/evaluationserver*. Pro stáhnutí a rozbalení zdrojových kódů je tedy nutno zadat příkaz

```
wget https://github.com/Foowie/evaluationserver/tarball/master -O
evaluationserver.tar.gz
```

```
tar -xzf evaluationserver.tar.gz
```

Zdrojové soubory nyní nalezneme v nově vytvořené složce.

8.1.3 Stažení pomocí verzovacího systému GIT

Nejnovější verzi je také možno získat pomocí verzovacího systému GIT. Pro stažení je nutno nejprve nainstalovat GIT klienta. Před instalací je však nutno se přepnout na uživatele, jež má oprávnění instalovat programy, následně je možno GIT klienta nainstalovat.

```
su root
```

```
nyní zadejte heslo uživatele root
```

```
apt-get install git
```

Instalace vás vyzve k potvrzení instalace potřebných balíků, tu potvrďte. Po úspěšné instalaci je možno získat celý repozitář se zdrojovými kódy pomocí příkazu

```
git clone https://github.com/Foowie/evaluationserver.git
```

Zdrojové kódy lze nyní nalézt ve složce *evaluationserver*.

8.1.4 Struktura zdrojových souborů

Po získání zdrojových kódů obsahuje hlavní složka podadresáře *server*, *tracer* a *web*. Ve složce *server* se nachází zdrojové kódy vyhodnocovacího serveru. Složka *tracer* obsahuje zdrojové kódy sandboxu, včetně testů a kostry vyhodnocovacího programu. Ve složce web se nachází soubory potřebné pro běh webového serveru. Mimo jiné se v hlavní složce také nachází soubor *database.mysq.sql* pro vytvoření struktury databáze a soubor *build.sh* pomocí kterého se provádí kompilace všech zdrojových kódů.

Po kompilaci přibude také složka *dist*, jež obsahuje připravené soubory systému k instalaci.

8.2 Kompilace zdrojových kódů

Před instalací je nutno nejprve zdrojové kódy zkompilovat. Pro kompilaci je potřeba nainstalovat programy, jež jsou nutné pro úspěšnou kompilaci vyhodnocovacího serveru. Nejprve je však nutno se přepnout na uživatele, jež má právo instalovat programy do operačního systému. To provedeme opět příkazem

```
su
```

```
nyní zadejte heslo uživatele root
```

Nyní bude potřeba nainstalovat programy ant a JDK. Ant je program, jež je využíván pro automatické sestavování softwarových aplikací. JDK je v systému Debian přítomen jako balíček *openjdk-6-jdk*. Pro instalaci těchto programů je tedy potřeba spustit následující příkazy:

```
apt-get install openjdk-6-jdk
```

```
apt-get install ant
```

Instalace bude vyžadovat instalaci dodatečných balíčků. Po instalaci potřebných programů nyní můžeme sestavit samotný vyhodnocovací server.

Po získání zdrojových kódů je nutno přejít do jejich složky. V té by měl být přítomen soubor *build.sh*. Spuštěním tohoto souboru pomocí příkazu

```
./build.sh
```

je započat proces sestavení vyhodnocovacího serveru. Pokud vše dopadlo úspěšně a během zpracování nebyla zobrazena varovná hláška, lze v podsložce *dist* vidět složky *server*, *web* a soubory *database.mysql.sql* a *uninstall.sh*. Pro jistotu je vhodné projít text zobrazený během kompilace.

8.3 Instalace

Po sestavení vyhodnocovacího serveru ze zdrojových kódů je potřeba nainstalovat jednotlivé komponenty. Postupně bude instalována databáze, vyhodnocovací server a webová část. Jako aktuální adresář je nyní brán adresář *dist*, jež byl vytvořen během procesu sestavení.

8.3.1 MySQL databáze

Nyní je čas nainstalovat MySQL databázi. Před instalací je opět nutno přepnout se na uživatele, jež má pravomoci instalovat balíčky, poté nainstalujte samotný MySQL server.

```
su
nyní zadejte heslo uživatele root
apt-get install mysql-server
```

Během instalace vás systém vyzve k zadání hesla pro uživatele databáze root, toto heslo vyplňte a dobře si ho zapamatujte. Nyní nastává chvíle k vytvoření nové databáze. Jako název databáze je zde použit evaluationserver, avšak může být i jiný. Je však potřeba tento název také změnit v konfiguraci.

```
mysql --user=root --password
nyní zadejte heslo databázového uživatele root
> create database evaluationserver;
> quit;
```

Databáze je vytvořena a nyní je potřeba jí předpřipravit. V aktuální složce se nachází soubor *database.mysq.sql*, který obsahuje potřebnou strukturu databáze a počáteční data. Naplnění se provede následujícím příkazem, pokud jste v předchozím kroku zvolili jiný název databáze, je nutné ji změnit i zde.

```
mysql --user=root --password evaluationserver < database.mysql.sql
nyní zadejte heslo databázového uživatele root
```

Pokud během předchozích kroků nevznikly komplikace, je databáze vytvořena. Pro produkční režim je vhodné vytvořit v databázi pro vyhodnocovací server zvláštního uživatele. Návod lze nalézt například v nápovědě k MySQL[22].

8.3.2 Vyhodnocovací server

V této části bude popsána instalace vyhodnocovacího serveru. Pokud jste neprovedli kompilaci zdrojových kódů, je nutno nainstalovat programy ant a JRE. Toho lze docílit pomocí následujících příkazů:

```
su
nyní zadejte heslo uživatele root
apt-get install openjdk-6-jre
apt-get install ant
```

22 Dokumenaci k MySQL serveru lze nalézt na adrese http://dev.mysql.com/doc/.

Nyní přejdeme do složky *dist/server*, ve které spustíme instalaci serveru. Předpoklad úspěšné instalace je být přepnutý na uživatele, jež má pravomoci zapisovat do systémových složek.

```
cd server
./install.sh
```

Tímto příkazem se vytvoří všechny nezbytné složky a nakopírují se do nich potřebné soubory.

8.3.3 Webový server

Webový server je možno provozovat dvěma způsoby. Server dodávaný spolu s webovým frameworkem, nebo jako servletový kontejner třetí strany. Popsána bude první možnost, kdy bude zprovozněn server dodávaný spolu s webovým frameworkem. Jak nainstalovat aplikaci do servletového, nebo aplikačního kontejneru se lze dočíst na stránkách frameworku[23].

Nejprve je nutno do systému nainstalovat program pro rozbalení ZIP archivu a samotný Play framework 1.2.4. To lze provést pomocí následujících příkazů:

```
wget http://download.playframework.org/releases/play-1.2.4.zip
su
nyní zadejte heslo uživatele root
apt-get install unzip
unzip play-1.2.4.zip -d /usr/share/
ln -s /usr/share/play-1.2.4/play /usr/sbin/play
```

Po instalaci frameworku přejdeme do složky *dist/web*, ve které spustíme instalační script.

```
cd web
./install.sh
```

Nyní je webový server nainstalován a je provedena kontrola všech závislostí na ostatní moduly. Dále je vygenerován nový klíč pro šifrování uvnitř webového serveru.

8.4 Konfigurace

Poslední věcí před samotným spuštěním je editace všech konfiguračních souborů. Veškeré konfigurační soubory lze nalézt ve složce */etc/evaluationserver*.

8.4.1 Vyhodnocovací server

Základní konfigurační soubor se nachází v */etc/evaluationserver/server/config.properties*. V něm je nutno změnit hodnoty `DATABASE`, `USERNAME` a `PASSWORD` na název databáze,

23 Dokumentaci k nasazení projektu na produkční server lze nalézt na adrese http://www.playframework.org/documentation/1.2.4/deployment.

uživatelské jméno a heslo pro přihlášení do databáze. Ostatní položky lze nastavit volitelně dle požadavků. Pokud tedy bude název databáze *server*, uživatelské jméno *admin* a heslo *password123*, bude počátek konfiguračního souboru vypadat následovně:

```
datasource.url = jdbc:mysql://localhost:3306/server

datasource.username = admin

datasource.password = password123
```

Další nastavení modulu vyhodnocovacího serveru je možno provést v souboru */etc/evaluationserver/server/logging.properties*. Zde se nacházejí informace o možnostech logování provedených serverem. Jedná se o standardní soubor s nastavením logování pro programovací jazyk Java.

8.4.2 Webový server

U webového serveru je potřeba editovat soubor */etc/evaluationserver/web/application.conf*. V něm je na prvních řádcích také potřeba změnit hodnoty podobně, jako u vyhodnocovacího serveru. Tedy `DATABASE`, `USERNAME` a `PASSWORD`. Ostatní hodnoty mohou zůstat nezměněny. Po nastavení hodnot stejně jako u vyhodnocovacího serveru by měl vypadat počátek tohoto konfiguračního souboru následovně:

```
db.url=jdbc:mysql://localhost:3306/server?characterEncoding=utf8

db.user=admin

db.pass=password123
```

8.5 Spuštění

Vyhodnocovací server lze spustit dvěma způsoby. Jako běžný program, nebo jako službu. Pro běžné spuštění jsou přidány příkazy *evaluationserver-server* a *evaluationserver-web*. Díky nim lze vyhodnocovací server a webový server spustit odkudkoliv. Pro správnou funkci je však zatím nutno spouštět vyhodnocovací server jako uživatel v roli správce, jelikož potřebuje zapisovat data a logovat do systémových složek.

Druhý způsob spuštění, tedy jako služby, lze také provést velmi snadno. Spuštění se skládá z příkazů běžně používaných pro spuštění služeb.

```
su

nyní zadejte heslo uživatele root

/etc/init.d/evaluationserver-server start

/etc/init.d/evaluationserver-web start
```

Samotný start webového serveru není triviální záležitost a proto je nutné počkat několik desítek sekund do jeho inicializace. Události prováděné jednotlivými službami lze nalézt v logu, respektive v souborech složky */var/log/evaluationserver/*. K webu lze nyní přistoupit pomocí

webového prohlížeče na adrese *http://localhost:9000/*. Port je možné změnit v konfiguračním souboru webového serveru. V případě potřeby je možno zapnout šifrování komunikace. Postup je popsán v nápovědě Play frameworku[24].

Pro přihlášení do systému jsou vytvořeny dva účty. Administrátorský, s přihlašovacím jménem „admin“ a heslem „admin“ a účet soutěžícího s uživatelským jménem „contestant“ a heslem „contestant“. Tyto účty jsou pouze ukázkové a před reálným použitím by měly být změněny hesla.

Systém po instalaci obsahuje také jednu tréninkovou soutěž, jednu úlohu a jedno správné řešení úlohy. Po spuštění vyhodnocovacího serveru se pokusí toto řešení vyhodnotit. Pro kontrolu správnosti výsledku je v systému soubor, jež je zkompilován pro systém *x86*. V případě architektury *x64* je nutno v úloze změnit program pro kontrolu na jiný. Ten je možno zkompilovat ze zdrojových kódů, jež jsou umístěny v souboru *tracer/diff/longiff.c*.

Pro zastavení běhu služeb stačí spustit níže uvedené příkazy.

```
su
nyní zadejte heslo uživatele root
/etc/init.d/evaluationserver-server stop
/etc/init.d/evaluationserver-web stop
```

8.6 Odstranění

Pokud je potřeba vyhodnocovací server odstranit ze systému, stačí spustit soubor *uninstall.sh*, jež se nachází po sestavení ve složce *dist*. Ten odstraní veškeré soubory vyhodnocovacího serveru, včetně dočasných souborů a logů.

Zbývá pouze odstranit databázi, framework a nepotřebné balíčky.

24 Dokumentaci k nastavení šifrování lze nalézt na adrese http://www.playframework.org/documentation/1.2.4/production.

9 Testy

Po dokončení implementace vyhodnocovacího serveru byl vyzkoušen v několika prostředích pro otestování jeho správné funkčnosti. Tabulka 10 zobrazuje jejich seznam, v nichž byl vyhodnocovací server testován.

Typ počítače	**Procesor**	**Paměť**	**Operační systém**	**Kernel**
Asus M50V	Intel Core2 Duo P7350	4 Gb	Xubuntu 11.04	2.6.38-14-generic i686
Asus M50V	Intel Core2 Duo P7350	4 Gb	Xubuntu 11.10	2.6.38-8 generic x68_64
VirtualBox	Intel Core2 Duo P7350	1 Gb	Debian 6.0.4	2.6.32-5-686 i686
VirtualBox	Intel Core2 Duo P7350	1,5 Gb	Ubuntu 12.04[25]	3.2.0-23 generic i686
Asus X51R	Intel Celeron M 520	0,75 Gb	Xubuntu 11.10	3.0.0-17 generic i686
Asus X51R	Intel Celeron M 520	0,75 Gb	Xubuntu 11.10	2.6.38-8 generic x68_64

Tabulka 10: Seznam prostředí pro testování

Během zkoušek byla nalezena chyba, jež zabraňuje využití vyhodnocovacího serveru na 64bitových operačních systémech. Za krátký čas se tuto chybu podařilo odstranit. Jednalo se o změnu názvů registrů u operačních systémů s jádrem *x86_64*.

Při testování bylo vytvořeno několik uživatelů v roli soutěžícího, několik úloh a soutěží. Následně byla provedena odeslání řešení úloh. Veškeré tyto činnosti byly provedeny s úspěchem.

25 Bylo použito JDK ve verzi 1.7.0_04.

10 Závěr

V předchozích kapitolách byla popsána funkce vyhodnocovacího serveru. Následně byly prozkoumány již existující vyhodnocovací servery, které jsou používány pro trénování a pořádání soutěží.

Byl proveden návrh takovéhoto serveru, rozdělení do modulů a rozebrání jednotlivých problémů vzniklých při návrhu. Vyhodnocovací server byl naimplementován a tato implementace byla popsána. Na výsledném programu byly provedeny řady testů. Nakonec byl sepsán návod, jak tento server nainstalovat a spustit.

Implementovaný vyhodnocovací server je plně funkční, avšak je zde mnoho vlastností, jež je vhodné rozšířit.

Jedna z věcí, které je možno na serveru vylepšit, je přidání dalších programovacích jazyků, jež je možno serverem zpracovat. Jedná se například o programovací jazyky Java, Python, Pascal a další. Díky nim by řešitelé mohli využít jazyk, s nímž mají největší zkušenosti.

V případě rozšíření tohoto serveru by bylo vhodné přidat detailnější správu uživatelských rolí. Díky němu by bylo možné přiřadit určitému administrátorovi skupinu soutěžících, který by je mohl spravovat.

Pokud by byl tento systém využit v soutěžích, nebylo by na škodu přidat dokumentaci standardních knihoven jednotlivých programovacích jazyků, případně doimplementovat formulář pro odesílání dotazů. K těmto věcem je sice možno využít i jiné přístupy, avšak implementace do vyhodnocovacího by byla přehlednější.

Zajímavá vlastnost je editor kódu s kompilerem přímo na webových stránkách, jako je například u serveru Light OJ. Tento doplněk se ale hodí spíše pro veřejně dostupný web.

S rozšiřováním výpočetní techniky stoupá i počet programátorů, jež mají chuť řešit složité algoritmické problémy. Posledních pár let zájem o tyto vyhodnocovací servery stoupá[26] a tak čeká soutěže v programování slibná budoucnost.

26 Podle statistik serveru UVa Online Judge počet odeslaných řešení od roku 2008 neustále stoupá.

11 Literatura

[1] - Active record pattern. *Wikipedia*: *The free encyclopedia* [online]. Last modified on 21 March 2012 at 00:34 [2012-02-03]. Dostupné z: http://en.wikipedia.org/wiki/Active_record_pattern

[2] - BlockingQueue: Java™ Platform Standard Ed. 6. ORACLE CORPORATION. *Java SE Developer Documentation* [online]. Wed Nov 30 13:41:38 PST 2011 [cit. 2012-04-08]. Dostupné z: http://docs.oracle.com/javase/6/docs/api/java/util/concurrent/BlockingQueue.html

[3] - Builder Pattern: SKUPINA: Creational Patterns. *Objekty*: *Návrhové vzory (design patterns)* [online]. 16.06.2005, 19:04 [2012-04-08]. Dostupné z: http://objekty.vse.cz/Objekty/Vzory-Builder. Diplomová práce M. Dvořáka.

[4] - CodeChef Blog: Announcing Variable Time Limits Based on Programming Language. *CodeChef*: *Programming Community, Programming Contest Platform* [online]. April 1st, 2009 [cit. 201111-11]. Dostupné z: http://blog.codechef.com/2009/04/01/announcing-time-limits-based-on-programming-language/

[5] - CodeChef: About. *CodeChef*: *Programming Community, Programming Contest Platform* [online]. [2011-11-11]. Dostupné z: http://www.codechef.com/aboutus

[6] - Immutable object. *Wikipedia*: *The free encyclopedia* [online]. Last modified on 29 April 2012 at 15:03 [2012-05-01]. Dostupné z: http://en.wikipedia.org/wiki/Immutable_object

[7] - Light OJ: Country wise Volume Ranklist. *Jan's LightOJ* [online]. [2012-11-12]. Dostupné z: http://www.lightoj.com/volume_country_ranklist.php

[8] - Observer Pattern: SKUPINA: Behavioral Patterns. *Objekty*: *Návrhové vzory (design patterns)* [online]. 16.06.2005, 19:04 [2012-02-18]. Dostupné z: http://objekty.vse.cz/Objekty/Vzory-Observer. Diplomová práce M. Dvořáka.

[9] - Open(2): open/possibly create file. *Linux man page* [online]. [2012-05-01]. Dostupné z: http://linux.die.net/man/2/open

[10] - Part III. Core Technologies: The IoC container. *Spring Framework*: *Reference Documentation* [online]. [2012-04-08]. Dostupné z: http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/beans.html

[11] - Project Euler: Statistics. *Project Euler*: *A website dedicated to the fascinating world of mathematics and programming* [online]. [2011-11-12]. Dostupné z: http://projecteuler.net/statistics. Dostupné po přihlášení.

[12] - Ptrace(2): process trace. *Linux man page* [online]. [2012-05-01]. Dostupné z: http://linux.die.net/man/2/ptrace

[13] - SIGALRM. *Wikipedia*: *The free encyclopedia* [online]. Last modified on 10 January 2012 at 14:36 [2012-05-01]. Dostupné z: http://en.wikipedia.org/wiki/SIGALRM

[14] - SIGPROF. *Wikipedia*: *The free encyclopedia* [online]. Last modified on 10 January 2012 at 14:39 [2012-05-01]. Dostupné z: http://en.wikipedia.org/wiki/SIGPROF

[15] - SIGSEGV. *Wikipedia*: *The free encyclopedia* [online]. Last modified on 27 April 2012 at 04:28 [2012-05-01]. Dostupné z: http://en.wikipedia.org/wiki/SIGSEGV

[16] - SIGTRAP. *Wikipedia*: *The free encyclopedia* [online]. Last modified on 10 January 2012 at 14:40 [2012-05-01]. Dostupné z: http://en.wikipedia.org/wiki/SIGTRAP

[17] - SIGVTALRM. *Wikipedia*: *The free encyclopedia* [online]. Last modified on 10 January 2012 at 14:40 [2012-05-01]. Dostupné z: http://en.wikipedia.org/wiki/SIGVTALRM

[18] - SIGXFSZ. *Wikipedia*: *The free encyclopedia* [online]. Last modified on 10 January 2012 at 14:40 [2012-05-01]. Dostupné z: http://en.wikipedia.org/wiki/SIGXFSZ

[19] - SPOJ Community Forum: What is SPOJ's _purpose_?. *Sphere Online Judge* [online]. Fri Sep 03, 2004 23:39 [2011-11-12]. Dostupné z: http://www.spoj.pl/forum/viewtopic.php?f=6&t=133

[20] - The main concepts: The MVC application model. *Playframework*: *Documentation* [online]. [2012-02-03]. Dostupné z: http://www.playframework.org/documentation/1.2.4/main#mvc

[21] - UVa Online Judge: Browse Problems. *UVa Online Judge* [online]. [2011-11-11]. Dostupné z: http://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8

[22] - UVa Online Judge: Quick Submit. *UVa Online Judge* [online]. [2011-11-11]. Dostupné z: http://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=25. Dostupné po přihlášení.

[23] - UVa Online Judge: Submission specification. *UVa Online Judge* [online]. [2011-11-11]. Dostupné z: http://uva.onlinejudge.org/index.php?option=com_content&task=view&id=15&Itemid=30

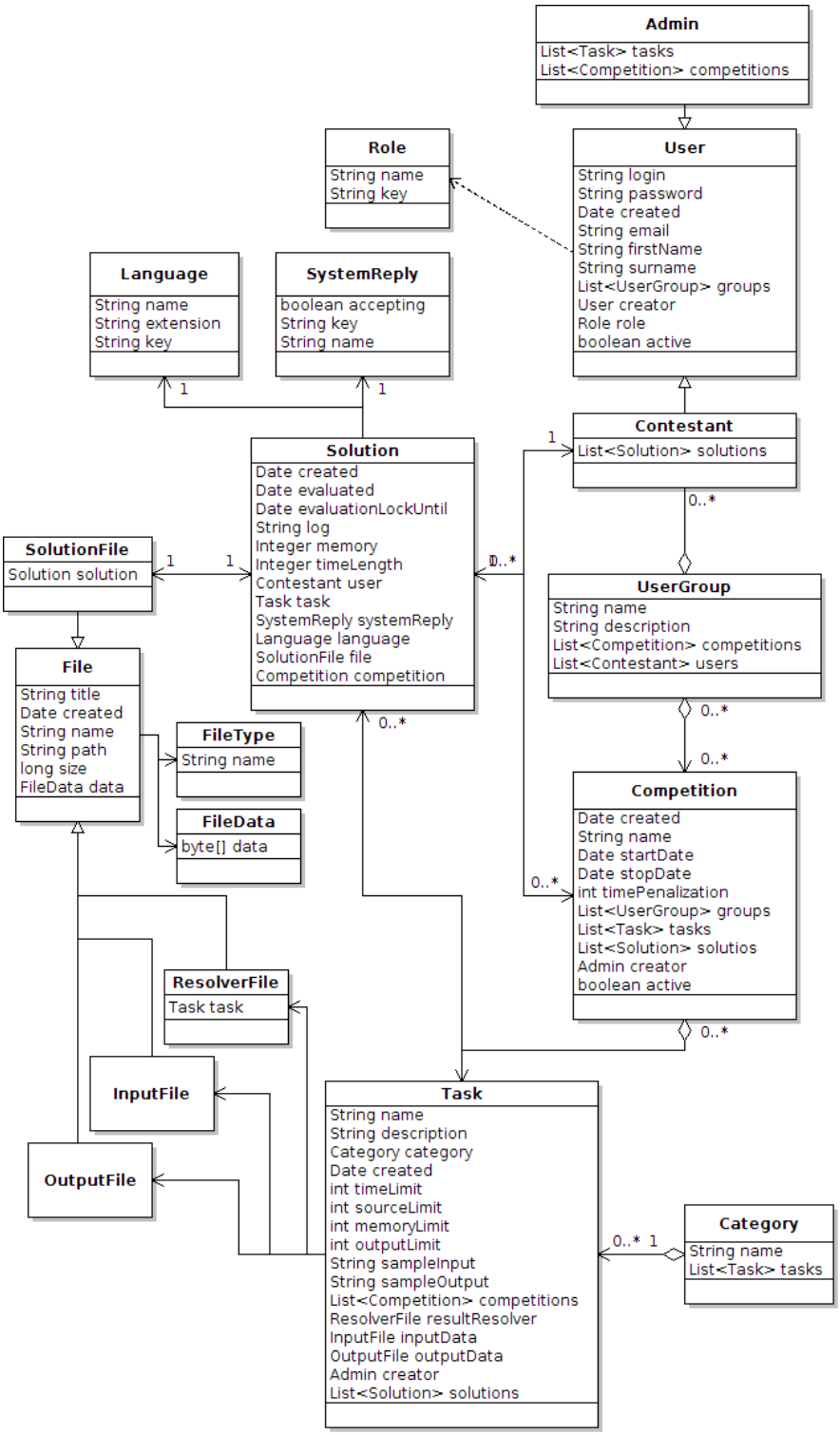
[24] - UVa Online Judge: Verdict information. *UVa Online Judge* [online]. [2011-11-11]. Dostupné z: http://uva.onlinejudge.org/index.php?option=com_content&task=view&id=16&Itemid=31

[25] - Validating HTTP data with Play. *Playframework*: *Documentation* [online]. [2012-02-18]. Dostupné z: http://www.playframework.org/documentation/1.2.4/validation

[26] - Write(2): to a file descriptor. *Linux man page* [online]. [2012-05-01]. Dostupné z: http://linux.die.net/man/2/write

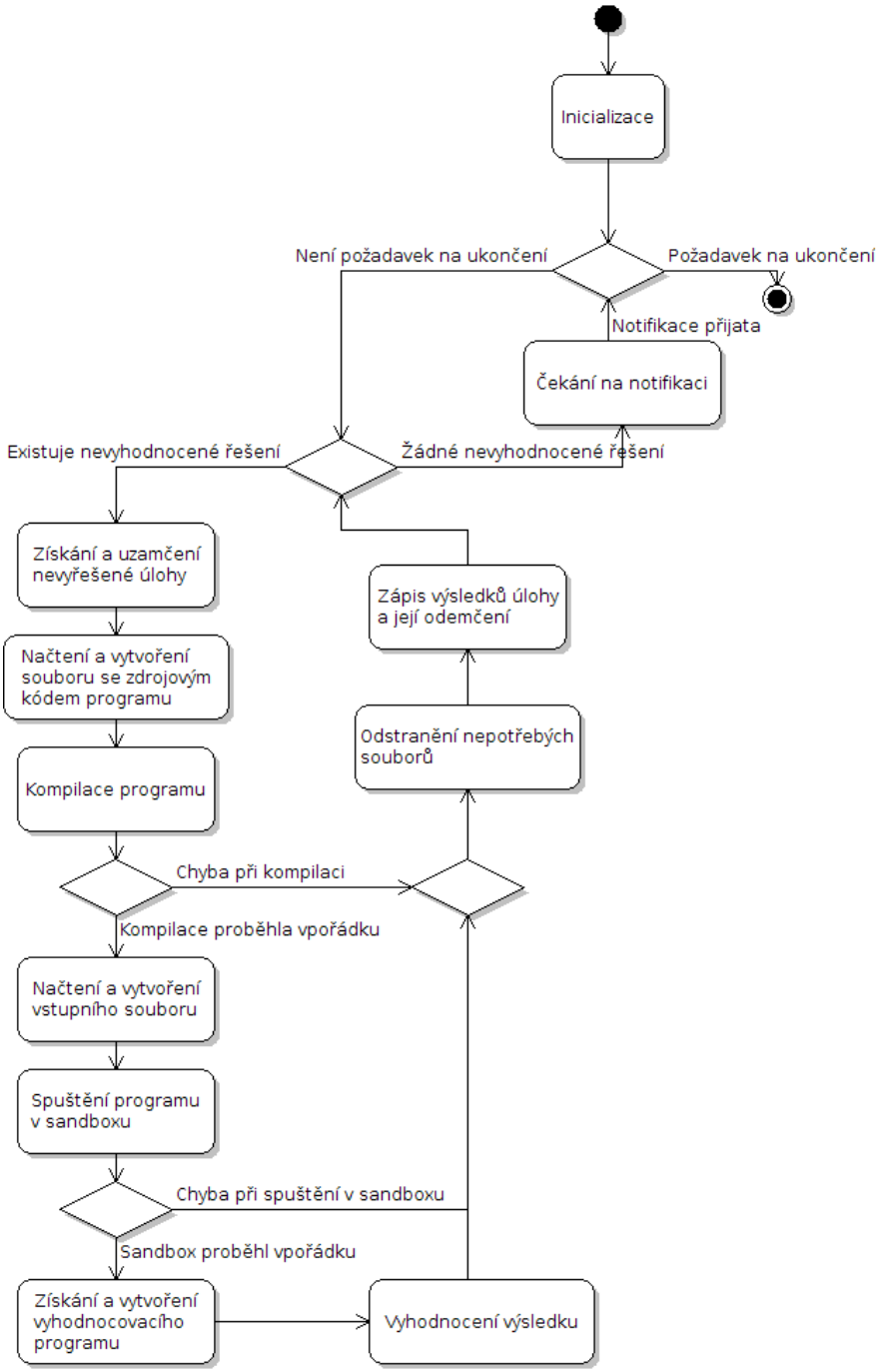
12 Seznam příloh

I. Třídní diagram entit webového serveru



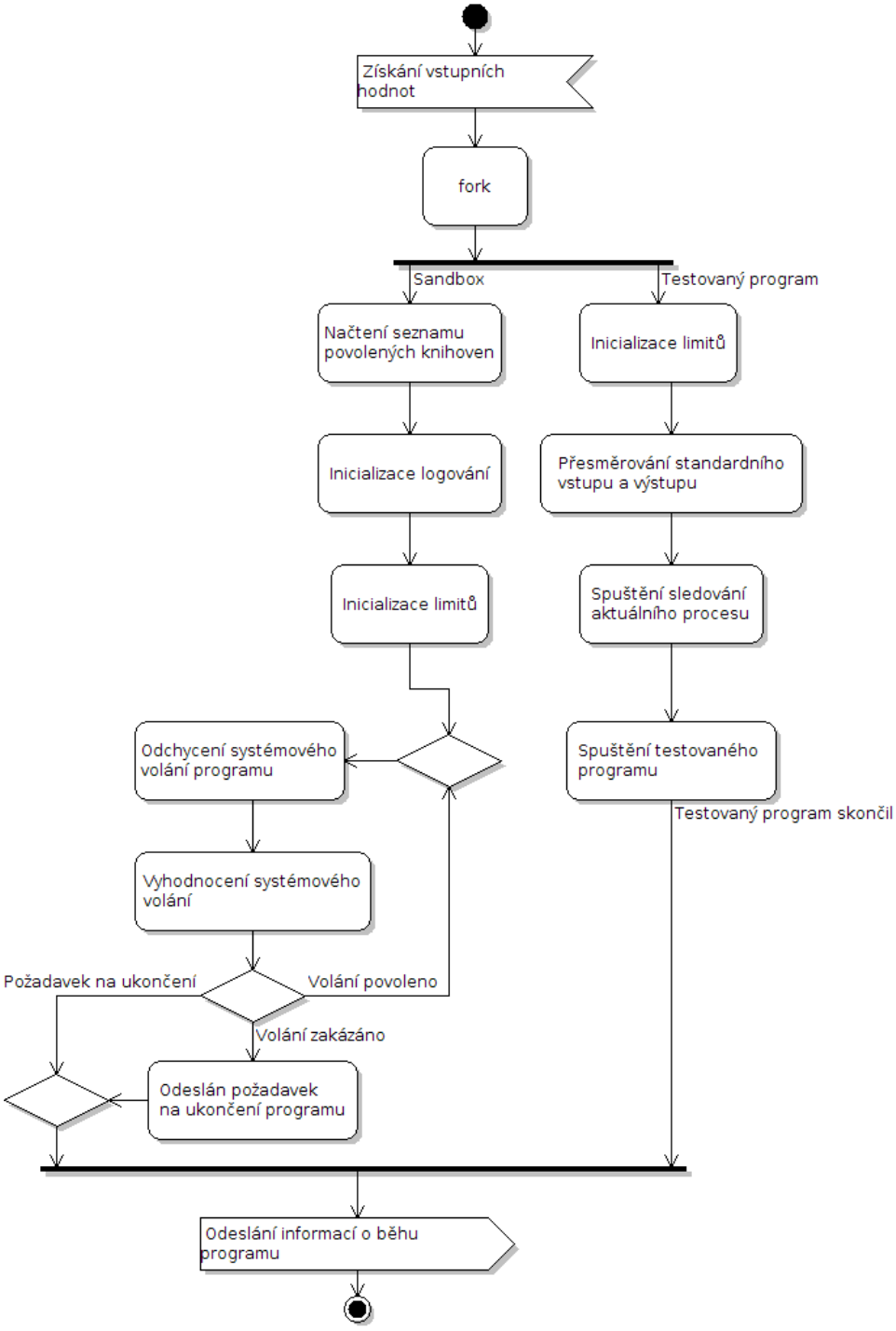
Obrázek 17: Třídní diagram entit webového serveru

II. Aktivitní diagram běhu vyhodnocovacího serveru



Obrázek 18: Aktivitní diagram běhu vyhodnocovacího serveru

III. Aktivitní diagram běhu sandboxu



Obrázek 19: Aktivitní diagram běhu sandboxu

IV. Specifikace softwarových požadavků

Dokument se specifikací softwarových požadavků je uložen na přiloženém CD pod názvem *Software requirement specification.pdf*.

Lze jej také nalézt jako přílohu k vázané diplomové práci za posledním číslovaným listem.