VŠB - Technická univerzita Ostrava
Fakulta Elektrotechniky a informatiky
Katedra informatiky

Automatizace hodnocení konfigurací v Distribuované virtuální laboratoři počítačových sítí

Ostrava 2009 Bc. Zdeněk Filipec

Zadání

List se zadáním

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 7. 5. 2009 ……..

Poděkování

Za vedení této práce a osobní přístup děkuji Ing. Petrovi Grygárkovi, Ph.D. Dále bych pak rád poděkoval celému vývojovému týmu Virtuální laboratoře za ochotnou a okamžitou pomoc při řešení problémů, které se vyskytly při vytváření systému automatizace konfigurací.

Abstrakt a klíčová slova

Abstrakt:

Cílem této práce je navrhnout a implementovat systém pro automatizované hodnocení konfigurací laboratorních prvků vytvářených ve Virtuální laboratoři. Projekt Virtuální laboratoře je nasazen do výukového procesu na Vysoké škole báňské v předmětech zabývajících se počítačovými sítěmi. Za účelem snadnějšího zadávání a efektivního hodnocení školních úloh a semestrálních projektů jsem vytvořil systém umožňující specifikovat různá zadání jednotlivých úloh pro skupiny studentů a umožňující automatizované hodnocení vyřešených úloh a projektů. Pedagogům používajících Virtuální laboratoř pro zadávání a zpracování semestrálních projektů ušetří vytvořený systém čas na opravu odevzdaných řešení.

Klíčová slova:

Virtuální síťová laboratoř, Virtlab, PHP

Abstract and key words

Abstract:

The point of this work is to design and implement the system for automated evaluation of the configurations of laboratory devices created in Virtual laboratory. The project of Virtual laboratory is engaged into educational process of the subjects dealing with computer networks on VSB – Technical University of Ostrava. I have created the system allowing to specify different assignments of the individual tasks to groups of students and allowing automated evaluation of solved tasks and semestral projects in order to make assignments easier and evaluations of the tasks and projects effective. Teachers using Virtual laboratory for assigning and processing semestral projects will save some time on evaluating student’s solutions by using this system.

Key words:

Virtual network laboratory, Virtlab, PHP

Seznam použitých symbolů a zkratek

AJAX – Asynchronous JavaScript and XML
CSS – Cascading Style Sheets – kaskádové styly
HTML – HyperText Markup Language – značkovací jazyk pro tvorbu internetových stránek
PHP – Personal Home Page – Hypertext Preprocesor – skriptovací jazyk
URL – Unique Resource Locator – jednoznačné určení zroje
XHTML – Extensible HyperText Markup Language – rozšiřitelný značkovací jazyk pro hypertext
XML – eXtensible Markup Language – rozšiřitelný značkovací jazyk

Obsah

Přílohy

A. Datový slovník a integritní omezení
B. Rozmístění zdrojových souborů
C. Obsah CD

Úvod

Distribuovaná virtuální laboratoř počítačových sítí umožňuje zpřístupnění sítových prvků laboratoře studentům přes Internet. Studenti si mohou rezervovat úlohu a v rezervovaném čase dovolí systém studentovi přistupovat a konfigurovat jednotlivé fyzické prvky rezervované úlohy.

V posledních letech roste počet studentů předmětů, které se zabývají počítačovými sítěmi na fakultě Elektrotechniky a informatiky. V současnosti se lektoři a pedagogové těchto předmětů potýkají z hlediska zadávání a hodnocení úloh s několika problémy:

a) Časová náročnost úloh – úkoly a témata k procvičení bývají obvykle časově náročné. Na cvičeních pak nezbývá čas pro samostatnou práci či pro práci na semestrálních projektech.
b) Množství studentů – uvážíme-li například, že v rámci jednoho cvičení je možno vyřešit mezní množství úloh (z důvodů omezeného počtu prvků a časové náročnosti), s rostoucím počtem studentů se také zvyšuje počet studentů pracujících na jedné úloze. Což vede k nepříjemným efektům jako například sdílení jednoho zařízení více studenty apod.
c) Oprava semestrálních projektů – počet zapsaných studentů například v předmětu Počítačové sítě je meziročně přibližně 300-350 studentů. Oprava takového množství odevzdaných projektů znamená pro pedagoga nemalou časovou zátěž.

Výše zmíněné problémy by se daly řešit využitím Virtuální laboratoře ve výuce. Aby však mohla být laboratoř efektivně používaná ve výukovém procesu, je potřeba systém Virtuální laboratoře rozšířit o možnost zadávání úloh studentům, diferenciaci jednotlivých zadání, a o možnost ověření správnosti řešení.

Tato diplomová práce si klade za cíl vytvořit systém automatizace hodnocení konfigurací, pomocí kterého se mají snížit časové nároky na kontrolu řešení jednotlivých úloh Virtuální laboratoře a tím také snížit počet studentů pracujících na jedné úloze.

Úkolem je tedy vytvořit softwarové dílo. Při vytváření díla je kladen důraz na použití objektového programování a na využití metod a prostředků softwarového inženýrství.

1. Seznámení s architekturou a implementací Virtuální laboratoře

1.1. Distribuovaná virtuální laboratoř (Virtlab)

Vznik projektu distribuované virtuální laboratoře (dále jen Virtuální laboratoř – Virtlab) byl iniciován v roce 2005. Původní tvář Virtlabu definoval Pavel Němec ve své diplomové práci [1]. Od této doby pracuje na projektu Virtuální laboratoře vývojový tým složený zejména z diplomantů katedry informatiky.

Virtlab pracuje na principu lokalit. Lokalitou se rozumí soubor technických a softwarových prostředků vytvářejících samostatnou funkční jednotku laboratoře. V rámci jedné lokality lze provádět všechny funkce, které systém nabízí. Distribuovanost Virtlabu spočívá v možnosti vytvořit více lokalit, které navzájem komunikují a spolupracují. Tímto lze rozšířit virtuální laboratoř do geograficky různých míst navzájem propojených sítí Internet, viz obr. 1.2. Síťové prvky jednotlivých lokalit jsou sdíleny s ostatními lokalitami.

Virtuální laboratoř umožňuje studentům katedry a dalším zájemcům přístup na laboratorní síťové prvky prostřednictvím sítě Internet. Uživatel si prostřednictvím webového prohlížeče vybere a rezervuje úlohu na informačním portále Virtlabu. Podle vybrané úlohy Virtlab vyhledá fyzické zařízení, která budou potřeba pro vytvoření topologie úlohy a tato zařízení rezervuje. Virtlab hledá vhodná zařízení ve všech lokalitách distribuované virtuální laboratoře. Jakmile se stane úloha aktivní, systém zajistí uživateli přístup na jednotlivé rezervované prvky prostřednictvím Java appletu běžícího ve webovém prohlížeči. Java applet simuluje řídící konzoly jednotlivých síťových prvků. Uživatelé tak reálně pracují s fyzickými prvky různých lokalit distribuované virtuální laboratoře.

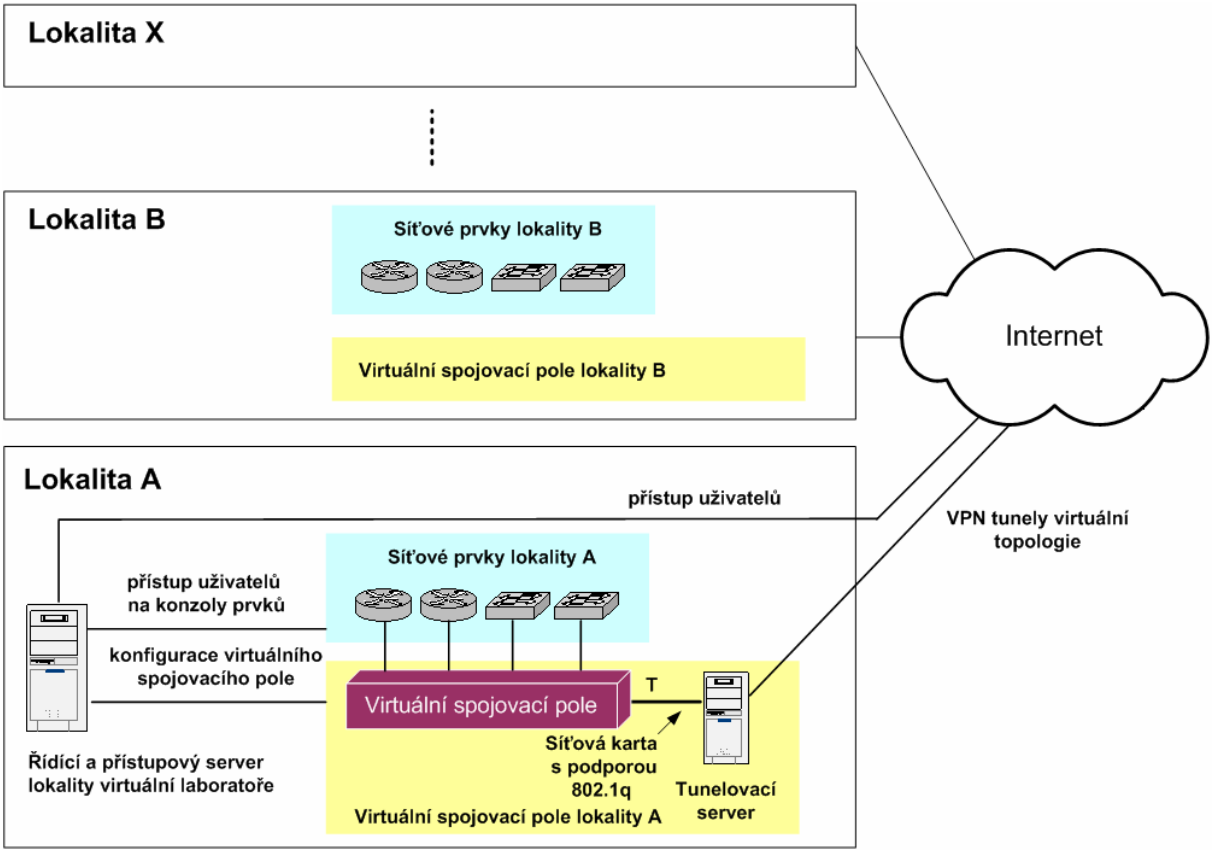
1.2. Architektura Virtlabu

Chod každé lokality zabezpečují softwarové moduly (servery), které zodpovídají za různé činnosti. Mezi nejdůležitější servery patří:

a) Řídící server – webové rozhranní umožňující uživateli práci s Virlabem. Slouží ke správě agendy uživatelů, úloh, rezervací atd. Uživatelé si pomocí něj mohou vybírat, rezervovat a vypracovávat úlohy. Za účelem provedení rezervace, komunikuje řídící sever se serverem rezervačním.
b) Rezervační server – zodpovídá za rezervaci fyzických síťových prvků jednotlivých lokalit. Poskytuje řídícímu serveru informace o dostupnosti síťových prvků a v případě potřeby je rezervuje.
c) Tunelovací server – Tunelovací servery zajišťují realizaci virtuální distribuované topologie přemosťováním provozu mezi rozhraními různých typů v lokalitách i mezi nimi prostřednictvím Internetu. Uživateli se jeví prvky rezervované úlohy jako kdyby byly

propojeny klasickým způsobem. I když ve skutečnosti se prvky nacházejí v různých lokalitách spojených sítí Internet.

d) Konzolový server – zajišťuje komunikaci uživatele s řídícími konzolami síťových prvků. Uživatelský vstup a výstup jsou směřovány skrze Java applet na konzolový server, který je dále přeposílá na sériové či telnetové konzoly síťových prvků. Pokud se nacházejí fyzické prvky rezervované úlohy ve vzdálených lokalitách, funguje také jako proxy server – přeposílá vstup a výstup na jiný konzolový server vzdálené lokality.



Obr. 1.2. – Architektura Virtlabu

Z hlediska této práce jsou nejdůležitějšími řídící (webový) server a konzolový server.

1.3. Webový server (řídící aplikace)

Základ současné verze řídící aplikace naprogramoval Jan Vavříček v diplomové práci [2] v roce 2007. Aplikace je implementována v programovacím jazyce PHP, datovou vrstvu tvoří databáze MySQL. Grafické rozhraní automatizace hodnocení konfigurací má být vytvořeno a zasazeno do stávající implementace řídící aplikace. Proto je žádoucí prostudovat a pochopit princip práce řídící aplikace a naučit se, jak do ní přidávat další dynamické webové stránky.

1.3.1. Implementace řídící aplikace

Řídící aplikace se skládá z dynamických webových stránek a několika PHP tříd obsahujících složitější funkce. Při implementaci řídící aplikace se využívá základních programovacích prostředků jazyka PHP, nebyla použita žádná externí řešení pro vývoj webových aplikací jako například frameworku MVC či mapování objektů na relační databáze atd. Jan Vavříček implementoval mimo jiné vlastní systém řešící směrování mezi stránkami a přístup na stránky podle práv uživatelů.

Směrování mezi stránkami

Jádro aplikace tvoří soubor index.php, na který jsou směřovány veškeré požadavky na zobrazení jednotlivých stránek. Index.php je jednoduchý controller, který na základě parametru page z URL požadavku rozhodne, který PHP soubor zahrne a zpracuje do odpovědi. Index.php nejprve vygeneruje společnou hlavičku stránky, dále pak na základě parametru page zahrne vybraný soubor a nakonec pak zahrne patičku stránky.

Parametr page je číselná konstanta, avšak vývojáři si nemusejí pamatovat číselnou hodnotu konstanty, místo toho vkládají do zdrojového kódu stránky jméno konstanty, která je v době generování obsahu stránky nahrazena číselnou hodnotou, viz. níže. Konstanty jsou definovány ve funkci Define_page_constants v souboru virtlabWeb.php.inc

Ověření jednotlivých požadavků uživatele se provádí pomocí session ID, které se neukládá do cookies na straně uživatele, avšak je předáváno v URL adresách požadavků. Proto musí vývojář do každého odkazu také vložit atribut sid nesoucí session ID otevřené relace.

Příklad 1.3.1.a – vložení nové dynamické stránky do řídící aplikace

---

1) úprava souboru virtlabWeb.php.inc – vytvoření konstanty stránky

```
public function Define_page_constants() {
        define('VC_PAGE_MAIN',             0);
        define('MY_PAGE_CONSTANT',         1);
}
```

2) vložení záznamu do souboru index.php

```
case MY_PAGE_CONSTANT:
        $body['body'] = $dir_virt . 'modul_testing/new_page.php';
        break;
```

V tomto místě říkáme controlleru index.php, který PHP soubor (new_page.php) má zahrnout v případě, že parametr page URL požadavku se rovná hodnotě konstanty MY_PAGE_CONSTANT.

3) vytvoření odkazu na novou stránku v těle jiné stránky

```
echo '<a href="index.php?page='. MY_PAGE_CONSTANT .'&amp;'. SID .'">
odkaz</a>';
```

Tento příkaz vygeneruje html odkaz na novou stránku. MY_PAGE_CONSTANT bude nahrazena hodnotou 1 a SID bude nahrazeno aktuálním session ID. Výsledný odkaz by vypadal například takto:

./index.php?page=**1**&sid=**9b6a65630252a54f80947d203477ba97**

Přístup na stránky na základě uživatelských práv

Ve Virtuální laboratoři může uživatel kromě základního uživatelského přístupu také nabýt práv: administrátor, správce úloh a tutor. Práva konkrétních uživatelů jsou uloženy v databázi. Po přihlášení přihlašovací skript řídící aplikace uloží do session proměnné $_SESSION[‘rights’] práva aktuálního uživatele.

Pro povolení či zakázání přístupu na určitou stránku podle práv uživatele lze využít funkce Autorization nacházející se v souboru virtlabWEB.php.inc. Funkce přijímá jediný parametr a tím je identifikátor stránky, u které se má ověřit, jestli má uživatel právo na její zobrazení. Parametr je textová proměnná a nijak nesouvisí s konstantami stránek využitých při směrování mezi stránkami. V příkladu 1.3.1.a je vytvořena stránka new_page.php. Příklad 1.3.1.b ukazuje, jak povolit přístup na tuto stránku pouze uživatelům s právy administrátora.

Příklad 1.3.1.b

1) úprava funkce Autorization v souboru virtlabWeb.php.inc

```
case 'another_page':
case 'my_new_page':
                if($_SESSION['rights']['admin']) return TRUE;
                else return FALSE;
```

- vložení větve case s identifikátorem nové stránky před podmínku kontroly práv administrátora.

2) vložení podmínky na začátek PHP souboru new_page.php

```
if(!$all->Authorization('my_new_page')) {
      //upozornění o nedostatečných právech
      return;
}//if-unauthorized
```

- proměnná $all je ukazatelem na instanci třídy virtlabWeb

Podrobnější informace o implementaci řídící aplikace lze nalézt v již zmíněné diplomové práci Jana Vavříčka [2].

1.4. Konzolový server

Konzolový server je démon napsaný v programovacím jazyce C, který slouží ke komunikaci s řídícími konzolami síťových prvků Virtuální laboratoře.

Konzolový server poslouchá na portu 10000. Abychom mohli komunikovat se síťovým prvkem, musíme konzolovému serveru nejprve odeslat identifikátor zařízení, na které chceme přistoupit a dále pak informace o tom, kdo chce komunikovat. Konzolový server totiž před

zprostředkováním přístupu ověřuje, má-li žadatel v daném čase právo komunikovat s daným zařízením. Pro zprostředkování přístupu musí tedy odeslat žadatel informace v tomto formátu:

```
id_zařízení@lokalita\r\n
SID\r\n (session ID)
tutor_mode\r\n
id_rezervace@lokalita\r\n
```

V rámci jedné lokality je každé zařízení identifikováno unikátním textovým řetězcem. V rámci celé distribuované virtuální laboratoře je pak každé zařízení identifikováno svým id a dále pak názvem lokality, ve které se nachází.

Služby konzolového serveru využívají především Java applety, přes které uživatel konfiguruje síťové prvky rezervované úlohy. Aby byl povolen přístup na daný prvek, odesílá applet konzolovému serveru session ID (SID) relace přihlášeného uživatele, dále pak tutor mód, ve kterém má komunikace probíhat, standardně „off“ a nakonec identifikátor rezervace úlohy uživatele ve tvaru id_rezervace@lokalita. Ověření se také provádí na základě uživatelovy IP adresy.

V mém případě budu přistupovat ke konzolovému serveru z řídící aplikace, tedy pod IP adresou zařízení, na kterém je spuštěn webový server. Jakékoli požadavky z IP adresy webového serveru konzolový server autorizuje. Nemusím tedy odesílat dále SID a id rezervace. Jako mód přístupu použiji exclusive mód, který zajistí, že komunikace mnou generovaná se nebude posílat případným uživatelům připojeným skrze Java applet na stejné zařízení. Přístup ke konzolovému serveru z IP adresy řídícího serveru je speciální případ pro systémové účely.

2. Specifikace požadavků

Specifikace požadavků vznikla na základě konzultací s vedoucím práce a na základě potřeb vyplývajících z konzultací s pedagogy, kteří se zabývají výukou počítačových sítí a kteří ručně opravují projekty řešené prostřednictvím Virtuální laboratoře.

2.1. Případy užití, aktéři

Pomocí diagramů případů užití bych rád popsal, co vše má systém umožňovat a dále pak znázornil, které funkce smí používat vybraní aktéři.

Se systémem kontroly a hodnocení (dále jen „systém“) budou pracovat tři typy aktérů: uživatel, student a učitel. Student je specializací uživatele a učitel specializací studenta. Studentovi jsou přístupny funkce uživatele a samozřejmě studenta. Učitel může využívat všech funkcí přístupných aktérům uživatel, student a učitel.

2.1.1. Případy užití - učitel

V nynější verzi Virtuální laboratoře je úloha specifikována textovým a grafickým zadáním pro pochopení studentem a dále pak specifikací požadavků na dynamicky vyhledávaná laboratorní zařízení a informací, jak daná zařízení spojit (pro fyzického zapojení). Pro potřeby testování konfigurace je nutné ke tvorbě úloh přidat definování pravidel testování a pro potřeby variability zadání také tzv. parametrizace jednotlivých zadání. Pravidla testování říkají, jak ověřit správnost řešení úlohy. Parametrizace zadání znamená vložení parametrů umožňující diferenciaci zadání úlohy na základě např. IP adresování, výběru směrovacích protokolů apod. Tyto parametry nemají vliv na topologii, avšak ovlivňují způsob řešení a výsledek testování.

Učiteli bude tedy systém umožňovat následující funkce (vyplývající z diagramu 2.1.1.):

a) Vytvářet a spravovat testovací pravidla[1] – učitel může do systému vkládat, upravovat a mazat testovací pravidla. Testovací pravidla může vytvářet pro konkrétní úlohu nebo může vytvořit globální pravidlo, které lze použít pro definici testů pro jakoukoli úlohu. Při definici testovacích pravidel se používají proměnné[2].
b) Rozšířit úlohu o testování – pokud si učitel přeje otestovat odevzdávané řešení, musí nejprve definovat jednotlivé testy[3] k vybrané úloze. Testem se rozumí vybrané testovací pravidlo s určením, na kterém prvku topologie se pravidlo provede. Pokud byly použity proměnné v definici testovacího pravidla, při vytváření testu se dále vybere, kterými hodnotami budou nahrazeny. Hodnoty mohou pocházet z jednotlivých parametrizací (zadáno učitelem) nebo mohou být zadány studentem při testování. Při definování testu se dále určí, kolik bodů získají studenti, pokud bude test úspěšný.

[1] vysvětlení pojmu testovací pravidlo viz. kapitola 2.3. Terminologie
[2] vysvětlení pojmu proměnná v kontextu s testovacími pravidly viz. kapitola 2.3. Terminologie
[3] vysvětlení pojmu test viz. kapitola 2.3. Terminologie

c) Parametrizovat úlohu – učitel může vytvořit tzv. parametrizace[4]. Parametrizace diferencují zadání úlohy a umožňují tak, aby skupiny studentů nevypracovávaly úlohu podle zcela stejného zadání.
d) Specifikovat části vypracovávání úlohy (šablona vypracování) – systém umožňuje učiteli zadat úlohu tak, aby ji studenti vypracovávali po částech. Pro každou část může vybrat testy, které se provedou (a ověří tak splnění/nesplnění řešení dané části) a také může říct, co mají studenti přiložit k řešení (například výpis směrovací tabulky). U jednotlivých částí vypracování je určen datum a čas, od kdy do kdy mají studenti danou část vypracovat.
e) Otevřít úlohu – otevřením úlohy se rozumí vytvoření skupinek studentů pro šablonu vypracování. Skupince může ale nemusí být přiřazena parametrizace upřesňující a diferencující zadání úlohy. Před otevřením úlohy musí být nejprve vytvořena šablona vypracování s minimálně jednou částí vypracování.
f) Prohlížet řešení – učitel může prohlížet řešení, které studenti odevzdali. Dále pak může na základě výsledků testů a odevzdaných informací řešení hodnotit. Systém bude také učiteli umožňovat zobrazit si podrobnou zprávu testování o provedených/neprovedených (z důvodů systémové chyby) testech (obsahující případné chybové hlášení apod.)
g) Archivovat – řešení studentů a výsledky testů odevzdaných úloh lze archivovat.

[4] vysvětlení pojmu parametrizace viz. kapitola 2.3. Terminologie

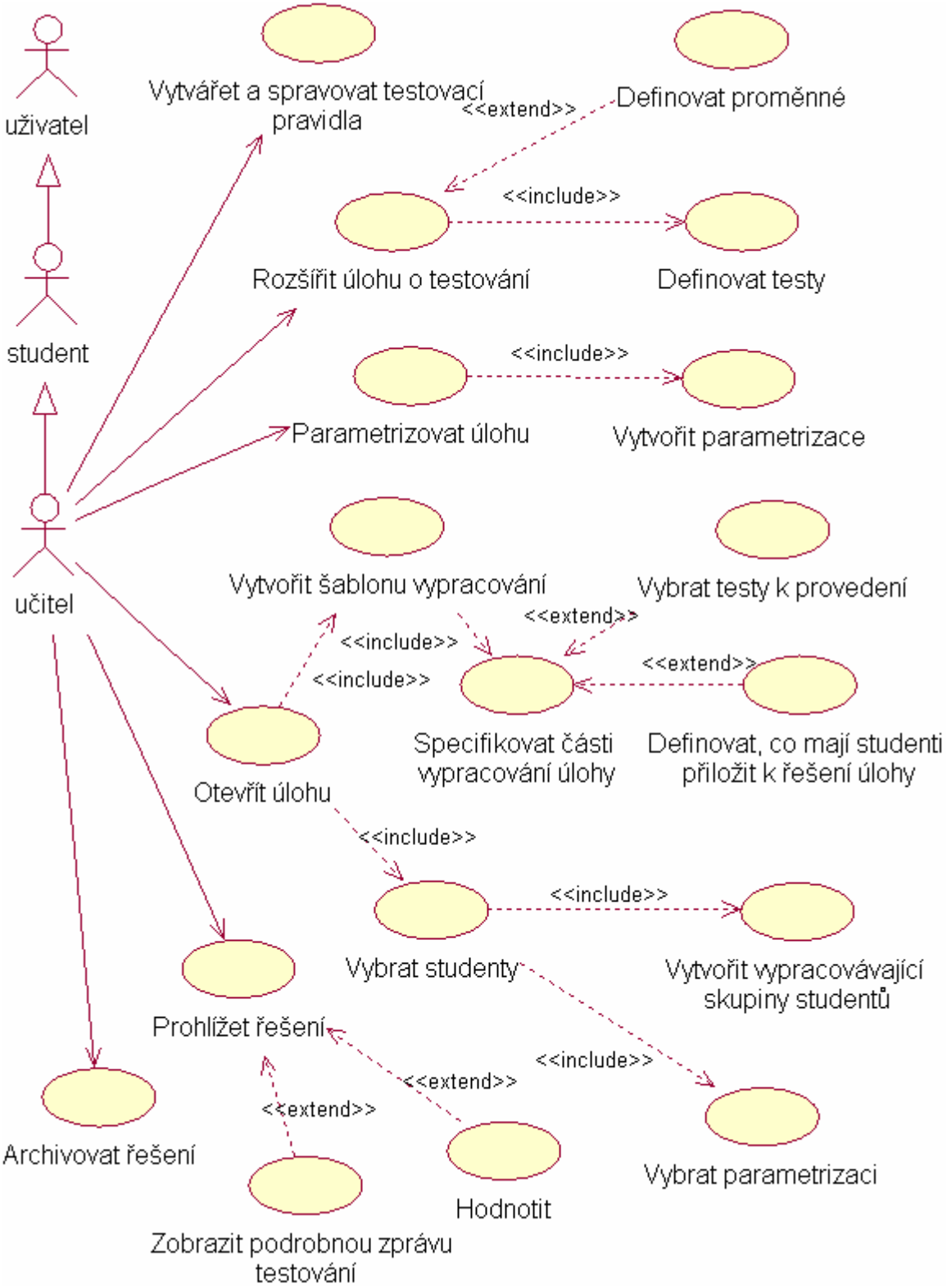


Diagram 2.1.1 – případy užití učitel

2.1.2. Případy užití – student, uživatel

Role uživatel představuje základní roli, ve které vystupují všichni aktéři používající řídící aplikaci Virtuální laboratoře. Všichni uživatelé systému mohou rezervovat úlohy, vypracovávat je a mohou provádět další základní funkce. V případě, že vyučující vybere uživatele do skupiny studentů přiřazené k některé úloze, získává navíc uživatel roli studenta a zpřístupňují se mu níže uvedené funkce. Uživatel si může rezervovat úlohu a pokud byl navíc zapsán k úloze, je mu dovoleno navíc jakožto studentovi úlohu vypracovat podle parametrizace, nechat si úlohu otestovat a odevzdat ji.

Aktér studenta přichází do interakce s následujícími případy užití:

1. Zobrazit zadané úlohy a jejich části vypracování – aby student věděl, které úlohy a jejich části má vypracovat, může si zobrazit seznam zadaných úloh. V případě, že učitel vybral pro skupinu, do které student patří, parametrizaci, bude studentovi zobrazeno také zadání parametrizace.
2. Vypracovat zadanou část úlohy – student může vypracovat část úlohy, která je zrovna otevřena. Pokud byla ke skupině, ve které se student nachází, přiřazena parametrizace, vypracovává student úlohu podle zadání parametrizace. V opačném případě podle obecného zadání úlohy.
3. Otestovat úlohu – pokud jsou k vypracovávané části úlohy přiřazeny testy, může si student nechat úlohu otestovat. V tom případě musí vyplnit testovací formulář[5], do kterého uvede hodnoty proměnných, které budou použity při testování. Systém bude studentovi umožňovat uložit si vyplněného hodnoty do příštího řešení stejné nebo jiné části úlohy.
4. Odevzdat vypracovanou část úlohy – odevzdání spočívá v provedení testů – viz bod b), ve vyplnění odevzdávacího formuláře[6] a v případě, že je student spokojen s výsledky testů, také v odeslání výsledků a odevzdávacího formuláře systému.
5. Zobrazit vypracované části úloh – student si může zobrazit seznam částí úloh, které již vypracoval a také se může podívat na své odevzdané řešení. V případě, že učitel ohodnotil odevzdané řešení, hodnocení je také zobrazeno studentovi.

[5] Vysvětlení pojmu testovací formulář viz. kapitola 2.3. Terminologie
[6] Vysvětlení pojmu odevzdávací formulář viz. kapitola 2.3. Terminologie

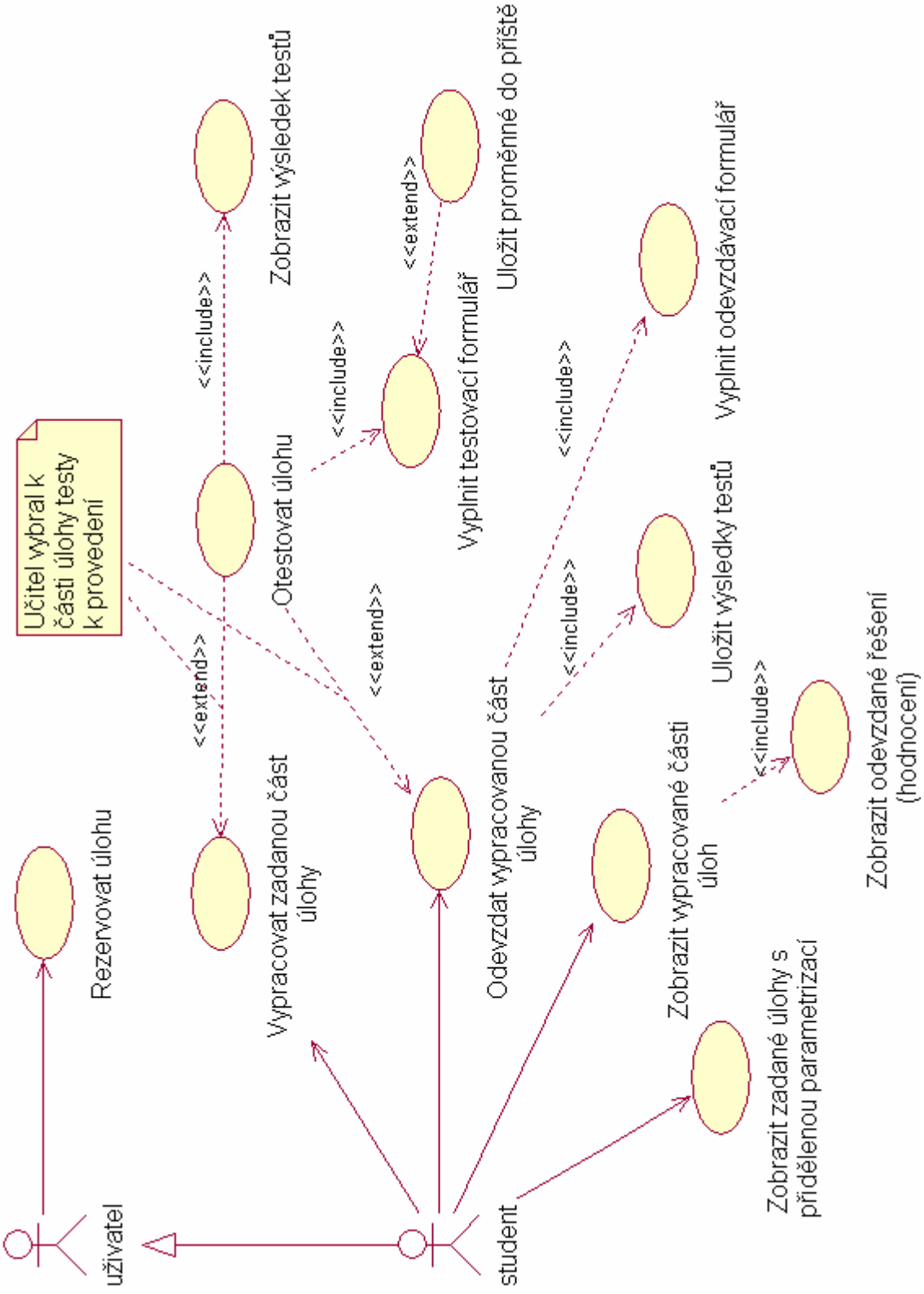


Diagram 2.1.2 – případy užití student, uživatel

2.2. Diagramy aktivit

Pomocí diagramů aktivit graficky znázorňuji, kterými aktivitami a v jakém pořadí budou aktéři procházet při používání různých funkcí systému.

Celkový pohled na systém popisující cestu od vytvoření testovacích pravidel, testů, parametrizací přes otevření úlohy studentům a jejího vypracování až k archivaci řešení vyjadřuje diagram aktivit 2.2.1.

2.2.1. Celkový pohled na systém

Poznámka s číslem diagramu u aktivity v obrázku 2.2.1. znamená, že danou aktivitu blíže popisuje jiný diagram aktivit. Před otevřením úlohy studentům vytvoří učitel nejprve testovací pravidla, poté definuje testy, volitelně parametrizace a jednolité části vypracování (šablona vypracování). Aby mohl student otevřenou úlohu, na kterou je zapsán, vypracovat, musí si ji nejprve rezervovat. Rezervace úloh je již implementována v řídící aplikaci. V době rezervace úlohy ji může vypracovat, otestovat a odevzdat. Tyto tři aktivity blíže popisuje diagram aktivit 2.2.7.

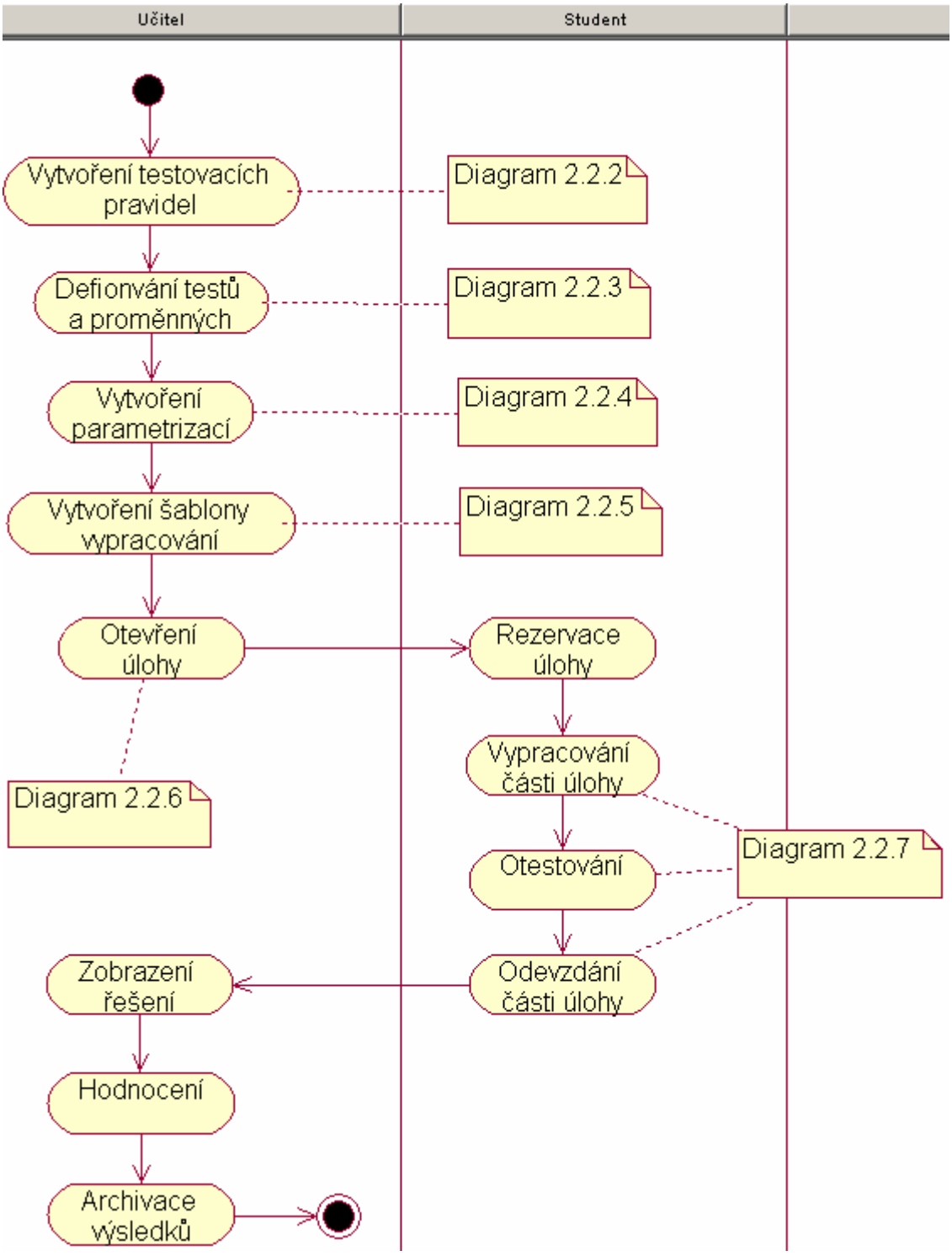


Diagram 2.2.1. – Celkový pohled na systém

2.2.2. Vytvoření testovacích pravidel

Aktivity prováděné při vytváření testovacího pravidla zobrazuje diagram 2.2.2. Testovací pravidla slouží k vytváření testů pro úlohu. Testovací pravidlo říká, jak se má testovat. A testy z nich vytvořené navíc určují, na kterém zařízení a se kterými proměnnými se má pravidlo vykonat. U testovacího pravidla nás zajímá název a typ testovacího pravidla, skript pravidla, koncová značka nebo timeout a nakonec název proměnných použitých v testovacím pravidle.

Typ pravidla říká, jestli může být pravidlo použito pro definici testů u všech úloh (typ globální) nebo bude pravidlo nabídnuto při vytváření testů pouze u konkrétní úlohy.

Skript pravidla tvoří sekvence příkazů, které budou při testování odeslány prostřednictvím konzolového serveru na síťový prvek, na kterém budou testy prováděny. V příkazech lze používat proměnné. Proměnné ve skriptu pravidla budou v době provádění pravidel nahrazeny hodnotami uživatelských nebo parametrizačních proměnných[7]. Hodnoty parametrizačních proměnných zadává učitel při vytváření parametrizací, hodnoty uživatelských proměnných zadávají studenti při vyplňování testovacího formuláře. Nahrazení proměnných se specifikuje při vytváření testů, viz kapitola 2.2.3. Jméno proměnné při použití ve skriptu se musí skládat pouze z písmen a čísel a musí být vloženo do dvojitých složených závorek.

Dále lze ve skriptu pravidla používat pauzy. Pauza umožňuje zastavit posílání příkazů pravidla na síťový prvek na určitou dobu. Pauz lze vhodně vyžít, potřebujeme-li pozastavit provádění skriptu z důvodů delšího vykonávání některých příkazů. Syntaxe pauzy je: {{wait(x)}}, kde za písmeno x přijde vložit celé číslo udávající počet vteřin udávajících délku pauzy.

Učitel vytvoří skript pravidla a následně nechá systém, aby ho analyzoval. Systém projde skript vložený učitelem a identifikuje veškeré proměnné a pauzy a zobrazí výsledek analýzy tvůrci pravidla. Jestliže byly použity proměnné, systém vyzve učitele k zadání uživatelsky příjemných názvů proměnných. V dalších částech systému budou všechny proměnné vystupovat pod tímto názvem, ne pod sekvencí znaků vložených do dvojitých závorek v textu skriptu.

Po odeslání všech příkazů cílovému prvku začne systém načítat odpověď. Tento výstup načítá do doby než za a) dosáhne v načítaném výstupu řetězce, který odpovídá koncové značce nebo za b) vyprší čas (time-out) pro čtení výstupu. Koncová značka nebo čas na čekání na výstup určí učitel při vytváření pravidla. Koncová značka je regulární výraz.

Posledním evidovaným údajem je vyhodnocovací regulární výraz. Po provedení testovacího pravidla bude ověřeno jestli výstup (odpověď) zařízení odpovídá regulárnímu výrazu. Pokud ano, je test vyhodnocen jako úspěšný. V opačném případě jako neúspěšný. Vyhodnocovací regulární výraz a koncová značka musí odpovídat syntaxi standardu POSIX Extended Regular Expression (ERE) [3].

Příklad testovacího pravidla (pouze pro demonstrační účely):

Název: ping

Typ: globální

Skript pravidla:

```
ping {{cilovaIP}}
{{wait(1)}}
```

Koncová značka – časová: 2 vteřiny.

[7] vysvětlení pojmů parametrizační a uživatelská proměnná viz. kapitola 2.3. Terminologie

Vyhodnocovací regulární výraz: !!!!!
Název proměnné {{cilovaIP}}: Cílová IP adresa

Ukázkové pravidlo sestává z jednoho příkazu – ping a jedné pauzy o délce 1 vteřina. V příkazu ping byla použita jedna proměnná, jež bude v systému vystupovat pod názvem Cílová IP adresa. Po odeslání pravidla zařízení, bude systém čekat 2 vteřiny (konec je určen timeoutem) po přijetí posledního znaku odpovědi a poté ověří jestli výstup odpovídá vyhodnocovacímu regulárnímu řetězci 5ti vykřičníků indikujících u zařízení Cisco úspěšnou odpověď na příkaz ping.

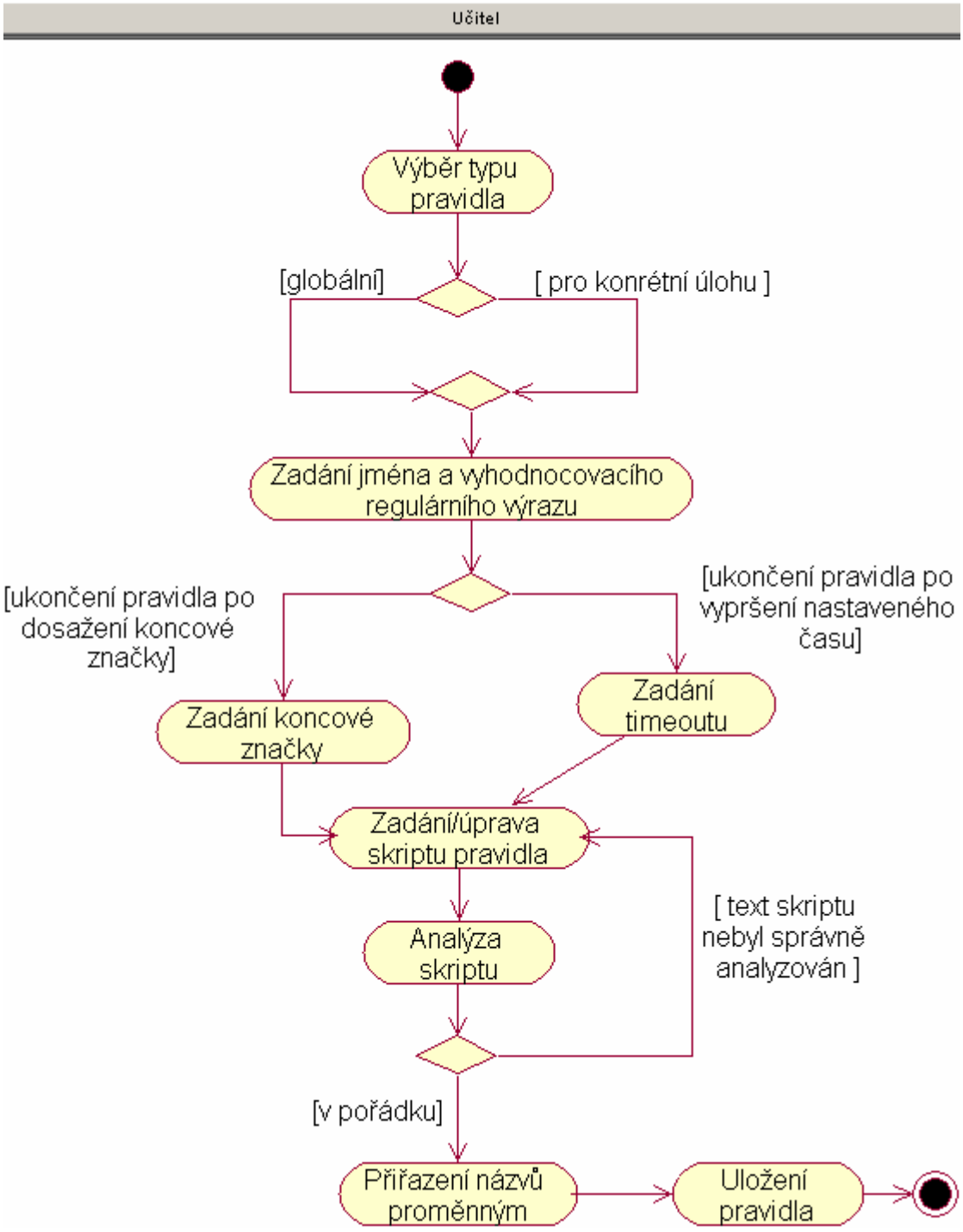


Diagram 2.2.2. – Tvorba testovacího pravidla

2.2.3. Definování testů a proměnných

Definování testů a proměnných zobrazuje diagram aktivit 2.2.3. Jednotlivé testy se definují pro konkrétní úlohy. Testem se rozumí vybrané testovací pravidlo s přiřazenými proměnnými. Při vytváření testů si může učitel vybírat ze seznamu globálních testovacích pravidel a konkrétních pravidel pro danou úlohu. Pokud byly ve skriptu testovacího pravidla použity proměnné, v této fázi učitel určí, kterými proměnnými mají být proměnné v testovacím pravidle nahrazeny v době provádění testu. Učitel tedy musí vytvořit parametrizační nebo uživatelské proměnné pro úlohu, pro kterou definuje testy. Tyto proměnné poté systém dosazuje do testovacích pravidel. Takto může vzniknout z jednoho testovacího pravidla více testů – každý test s jinými proměnnými. Navíc také u vytvářeného testu učitel vybírá, na kterém síťovém zařízení se má test provést.

Nakonec pak učitel u testu nastavuje povinnost a počet bodů, které studenti získají, pokud je test úspěšně splněn. Všechny testy označené jako povinné pro vybranou část úlohy musejí být splněny, aby byl celkový výsledek testování vyhodnocen jako úspěšný.

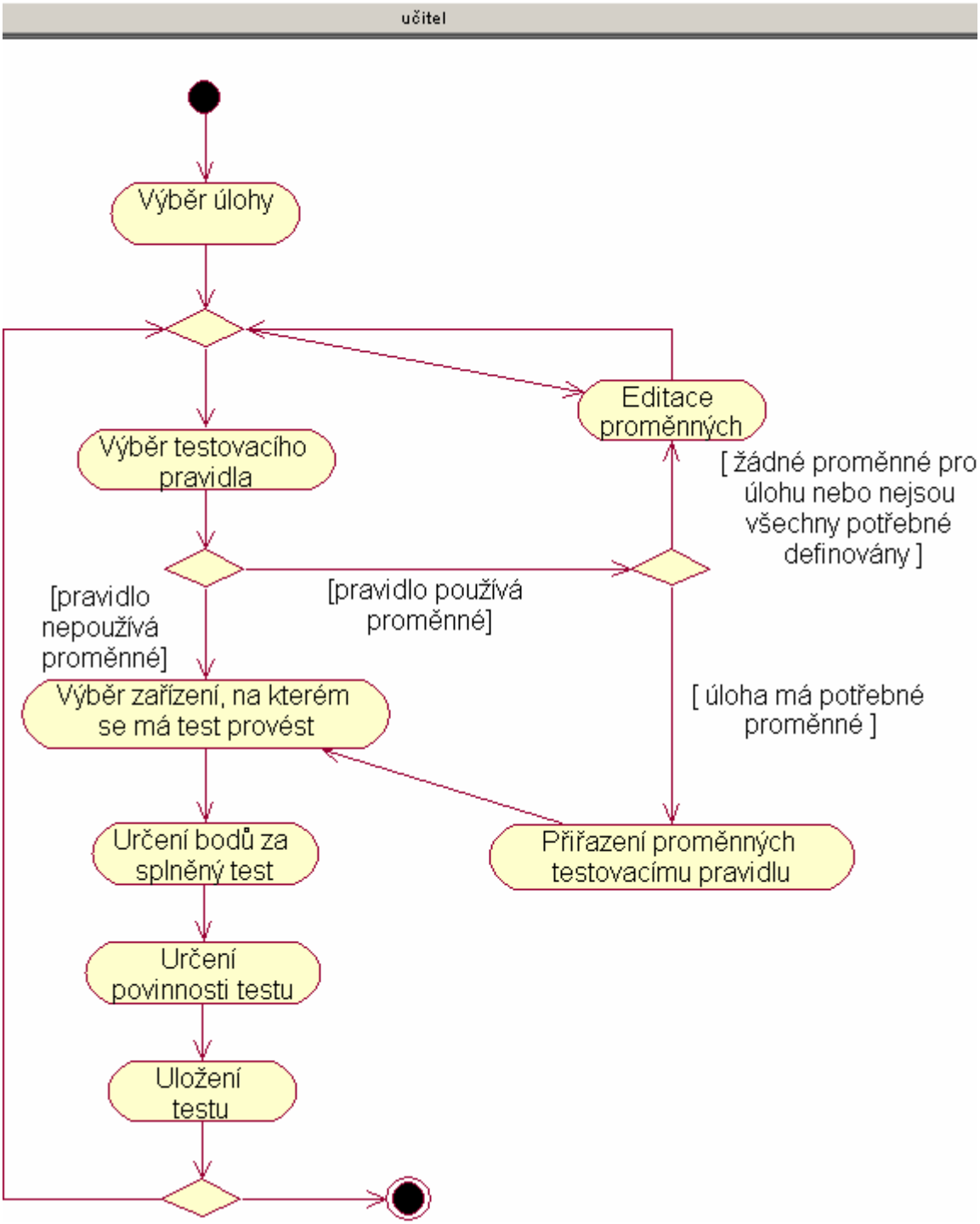


Diagram 2.2.3 – Definování testů

2.2.4. Vytvoření parametrizací

Parametrizace vytváří učitel pro vybranou úlohu. Parametrizace slouží k diferenciaci zadání pro jednotlivé skupiny studentů. Skládají se ze slovního popisu upřesňující zadání a z určení hodnot parametrizačních proměnných. Pokud učitel využil při definici testů pro vybranou úlohu parametrizační proměnné, při vytváření parametrizace jim vyplní hodnoty, které mají být použity v době provádění testů. Vytvoření parametrizace zobrazuje diagram 2.2.4.

Parametrizace mohou být vytvořeny, i když úloha nemá definovány žádné testy. Studenti pak budou úlohy vypracovávat podle popisu parametrizace, ale nebudou provedeny žádné testy. Popisem parametrizace může být například zadání rozsahu IP adres, který mají použít k adresaci topologie úlohy. Každá skupinka studentů tak získá jiné zadání.

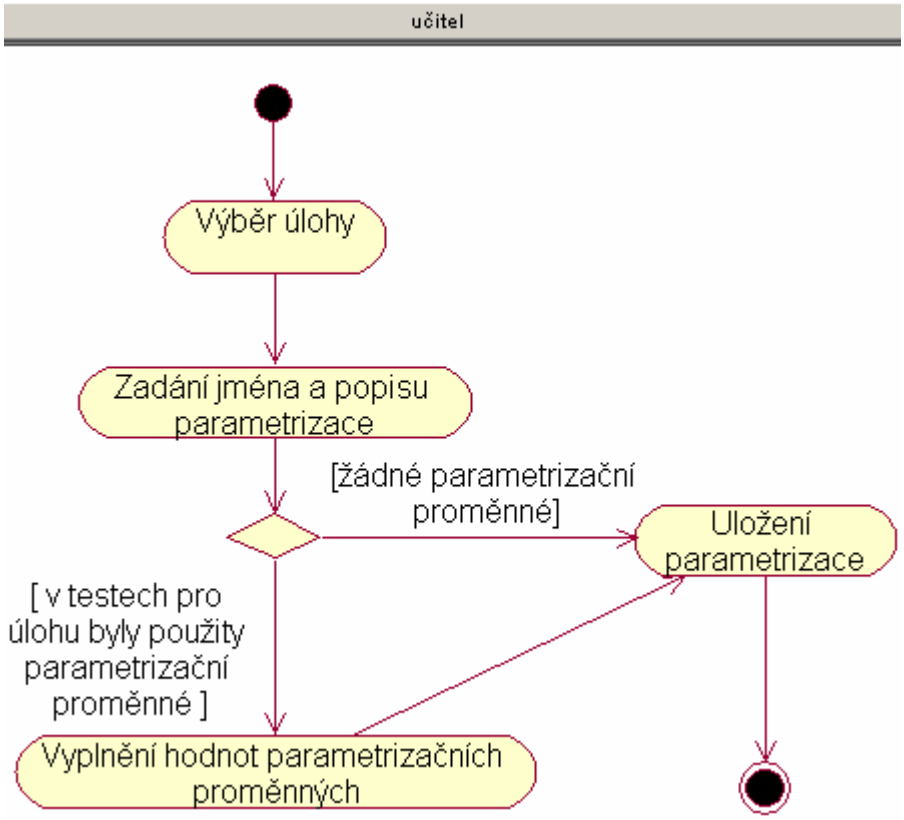


Diagram 2.2.4 – Vytvoření parametrizací

2.2.5. Vytvoření šablony vypracování

Úlohu nelze otevřít studentům k vypracování, dokud není vytvořena šablona vypracování s minimálně jednou částí vypracování.

Diagram 2.2.5. zobrazuje aktivity, kterými prochází učitel při vytváření šablony vypracování.

Úloha nemusí být vypracována studenty najednou. V případě rozsáhlejších úloh může učitel specifikovat více částí vypracování. Část vypracování je určena datem a časem začátku a konce, kdy má být studenty vypracována a odevzdána. Dále pak požadavky na přílohy k řešení – co mají studenti vyplnit do odevzdávacího formuláře. A nakonec výběrem testů, které se provedou při odevzdávání části.

Pro názornost uvedu ilustrační příklad. Mějme úlohu na procvičení dynamického směrování pomocí protokolu RIPv2, která se jmenuje Směrování s protokolem RIPv2. Zadání úlohy obsahuje základní informace o protokolu RIPv2 a obecné zadání. V jednotlivých parametrizacích pro skupiny studentů by mohl učitel specifikovat rozdílné rozsahy adres pro adresování stanic a síťových prvků. Tímto zajistí diferenciaci zadání pro odlišné skupiny. Učitel vytvoří šablonu vypracování a pojmenuje ji SPS 2009. V rámci této šablony vytvoří dvě části vypracování. U první části určí, že má být vypracována od 1.4 – 15.4.2009, nevybere pro ni žádné testy, ale v požadavcích na přílohy vyzve studenty k přiložení popisu adresování a směrovacích tabulek směrovačů v topologii. U druhé části bude chtít, aby byla vypracována a odevzdána od 1.5 – 15.5.2009. Tentokráte vybere testy, které ověřují konektivitu jednotlivých zařízení na základě příkazu ping a nebude požadovat žádné přílohy k řešení.

Aby mohli studenti vypracovat obě části, musí si nejprve rezervovat úlohu Směrování s protokolem RIPv2 na datum od 1.4. do 15.4.2009, v době rezervace úlohu vypracovat a k odevzdání části přiložit směrovací tabulky směrovačů. Následně pak na dobu od 1.5. do 15.5.2009 znova rezervovat úlohu Směrování s protokolem RIPv2, vypracovat ji a tentokráte nechat topologii otestovat testy, které učitel vybral pro druhou část vypracování, a úlohu odevzdat.

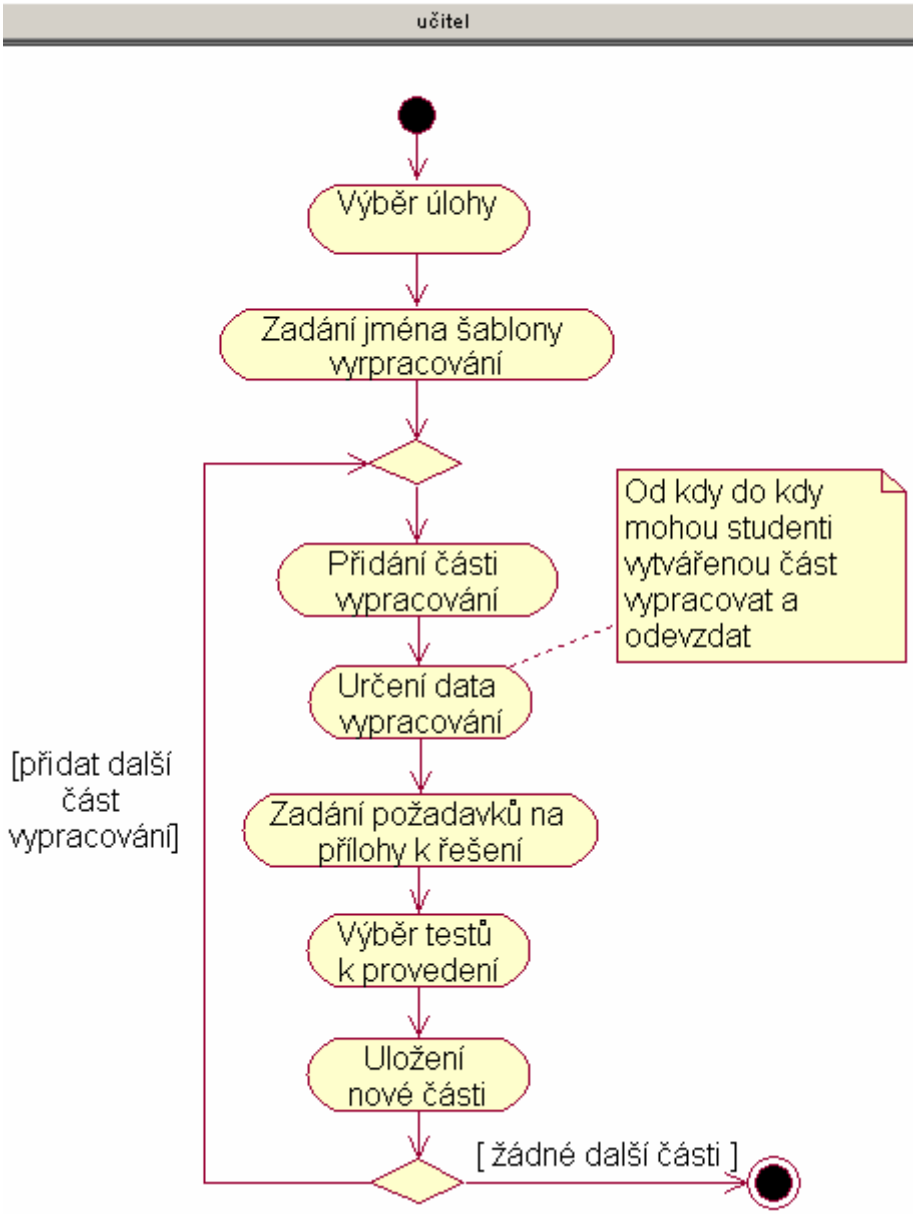


Diagram 2.2.5. – Šablona vypracování

2.2.6. Otevření úlohy

Otevření úlohy prakticky znamená definování skupin studentů k šabloně vypracování. Každé skupině lze volitelně přiřadit jednu parametrizaci, podle které by měla úlohu řešit. Proces otevírání úlohy zobrazuje diagram 2.2.6. Jakmile jsou přiřazeni skupiny studentů k šabloně vypracování, stává se úloha, pro kterou byla šablona vypracování vytvořena, otevřena. Zapsaní studenti tak mohou vypracovávat a odevzdávat jednotlivé části vypracování dané šablony.

Systém bude umožňovat dva způsoby vytváření skupin studentů:

a) ruční vytváření – učitel vybere ručně pomocí formuláře studenty do skupiny a přeje-li si, aby skupina vypracovávala zadání podle parametrizace, přiřadí skupině parametrizaci.
b) automatizované vytváření pomocí souboru – v případě velkého počtu studentů pro jednu úlohu může učitel využít automatizované vytváření. Vloží do systému párovací soubor obsahující informace o vytvářených skupinách studentů s přiřazenými parametrizacemi.

Formát řádku párovacího souboru:

```
id_parametrizace;id_uzivatele,id_uzivatele,id_uzivatele
```

Na každém řádku souboru se nachází informace o jedné skupině. Identifikátor parametrizace určuje parametrizaci pro skupinu. Identifikátor uživatele je přihlašovací jméno uživatele používané k přihlášení do řídící aplikace. Identifikátor parametrizace se od seznamu uživatelů odděluje znakem středník (**;**), přihlašovací jména uživatelů jsou od sebe odděleny znakem čárka (**,**).

Systém bude umožňovat exportovat v textovém formátu názvy existujících parametrizací a jejich identifikátorů. Pokud učitel nechce přiřadit skupině parametrizaci, jako id_parametrizace uvede identifikátor -1.

Po odeslání párovacího soubor systém identifikuje skupiny studentů s přiřazenými parametrizacemi a informuje učitele o výsledku analýzy (jaké skupiny nalezl). Pokud učitel souhlasí s výsledkem, potvrdí vytvoření skupin s parametrizacemi.

Příklad párovacího souboru:

```
5;jana,pepa
-1;petra,lukas,alena
```

Pomocí tohoto párovacího souboru by byly vytvořeny dvě skupiny studentů, v první skupině by byli uživatelé jana a pepa a vypracovávali by úlohu podle parametrizace 5. Ve druhé skupině by byli uživatelé petra, lukas a alena a úlohu by vypracovávali pouze podle obecného zadání (nebyla jim přiřazena parametrizace).

V okamžiku zapsání uživatele do skupiny vstupuje uživatel do role studenta a zpřístupňují se mu příslušné funkce systému.

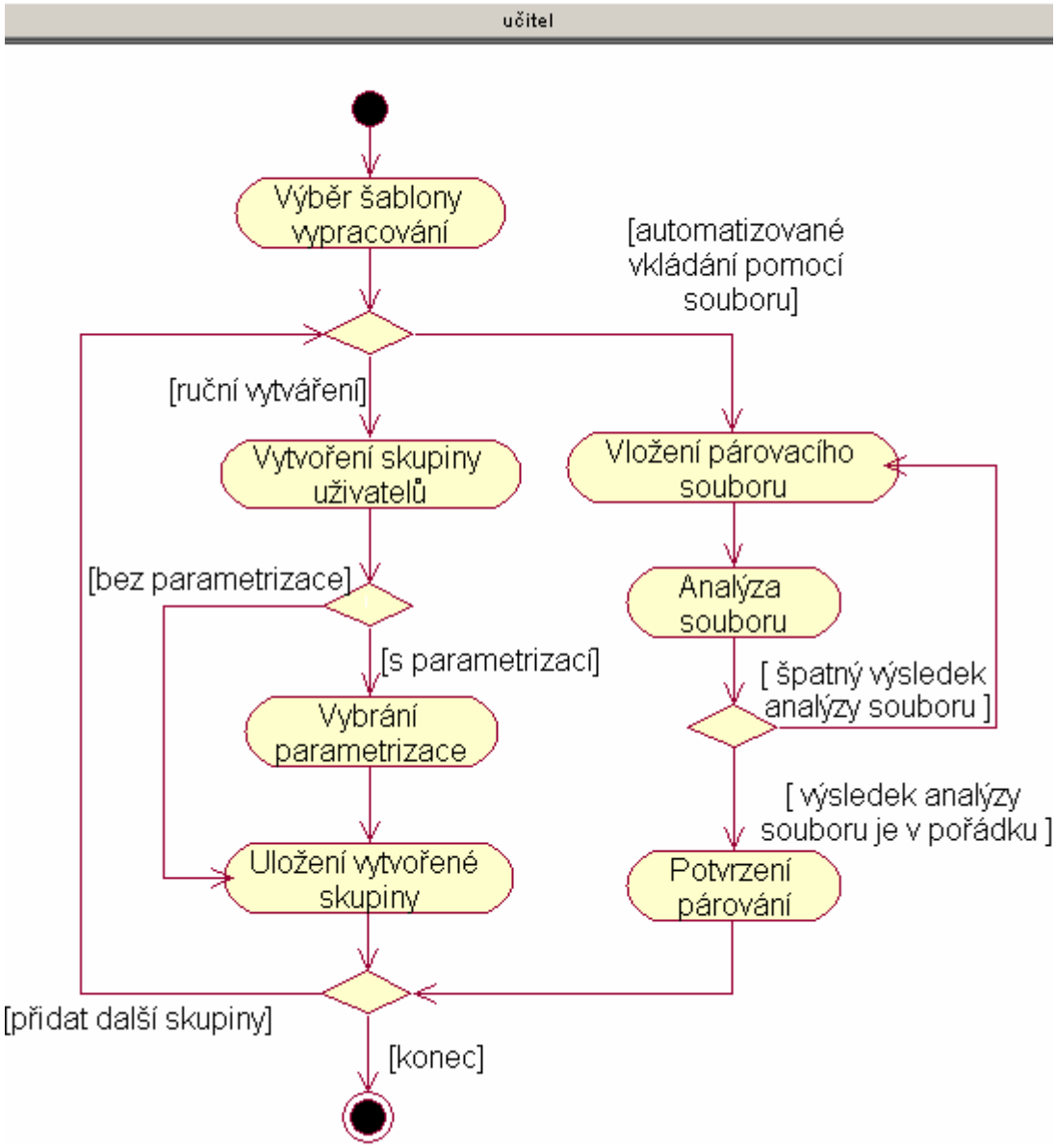


Diagram 2.2.6. – Otevření úlohy

2.2.7. Testování a odevzdání části úlohy

Student po vyřešení úlohy (např. po konfiguraci prvků) může odevzdat řešení. Pokud vyučující vybral pro řešenou část testy, systém otestuje nakonfigurovanou topologii. Pokud učitel použil ve vybraných testech alespoň jednu uživatelskou proměnnou, bude student vyzván k vyplnění odpovídajících hodnot do testovacího formuláře. Jestli pro odevzdávanou část vypracování nejsou specifikovány žádné testy, bude studentovi zobrazen přímo odevzdávací formulář. Ten mu bude zobrazen i po provedení testů spolu s jejich výsledky. V tomto kroku se může student rozhodnout pro vyplnění odevzdávacího formuláře a odevzdání úlohy nebo pro úpravu řešení úlohy na základě výsledků testů. Do odevzdávacího formuláře vkládá student informace, které žádá vyučující jakožto požadavky na přílohy k řešení (definováno při vytváření části vypracování). Po odevzdání budou výsledky testů a informace z odevzdávacího formuláře uloženy do databáze a budou přístupny jak studentům tak učiteli. Pro potřeby učitele bude navíc vygenerována a uložena podrobná zpráva o testování obsahující systémové výpisy a případné chyby vzniklé při provádění testů.

Pokud danou část vypracování řeší skupina studentů, může nechat úlohu otestovat a řešení odevzdat jakýkoli student ze skupiny. Řešení úlohy se odevzdává za celou skupinu. Systém bude umožňovat odevzdat pouze jedno řešení ke každé části vypracování za každou skupinu.

Systém bude dále umožňovat studentům uložit si vyplněné hodnoty uživatelských proměnných testovacího formuláře. Pokud má úloha vytvořeno více částí vypracování, nemusí studenti při řešení každé části vypracování vyplňovat znova uživatelské proměnné, pokud si je uložili při vypracovávání části předešlé. Uložit proměnné si mohou i když úlohu neodevzdají a přejí si ji vypracovat v jiný čas. Dále je pak testovací formuláře ukládán do session, student se tak může volně pohybovat mezi stránkami řídící aplikace a nemusí testovací formulář vyplňovat vždy, když chce provést testování.

Výsledek testování obsahuje datum a čas začátku a konce testování, statistiku počtu provedených/neprovedených a úspěšně vyhodnocených/nevyhodnocených testů a dále pak seznam všech provedených testů s jejich detaily. Test nemusí být proveden z důvodů systémové chyby. V seznamu testů bude uvedeno jméno testu, zařízení na kterém byl proveden, skript testu (proměnné budou nahrazeny hodnotami), počet bodů, které student dostane, jestliže byl test vyhodnocen jako úspěšný a nakonec bude uvedeno, zda byl test povinný nebo ne. U neprovedených testů bude zobrazeno chybové hlášení.

Učitel bude hodnotit řešení studentů na základě informací z odevzdávacího formuláře a na základě výsledků testování. V případě, že některé testy nebudou provedeny či se vyskytnou jiné nesrovnalosti, může si učitel zobrazit podrobnou zprávu testování. Ta bude navíc obsahovat u každého provedeného testu oproti výsledku testování pro studenta tyto informace:

a) jméno určující fyzický prvek v rámci distribuované virtuální laboratoře, na kterém byl test proveden
b) typ a hodnotu koncové značky
c) regulární výraz vyhodnocení testu
d) výstup, který vygenerovalo zařízení po provedení skriptu testu

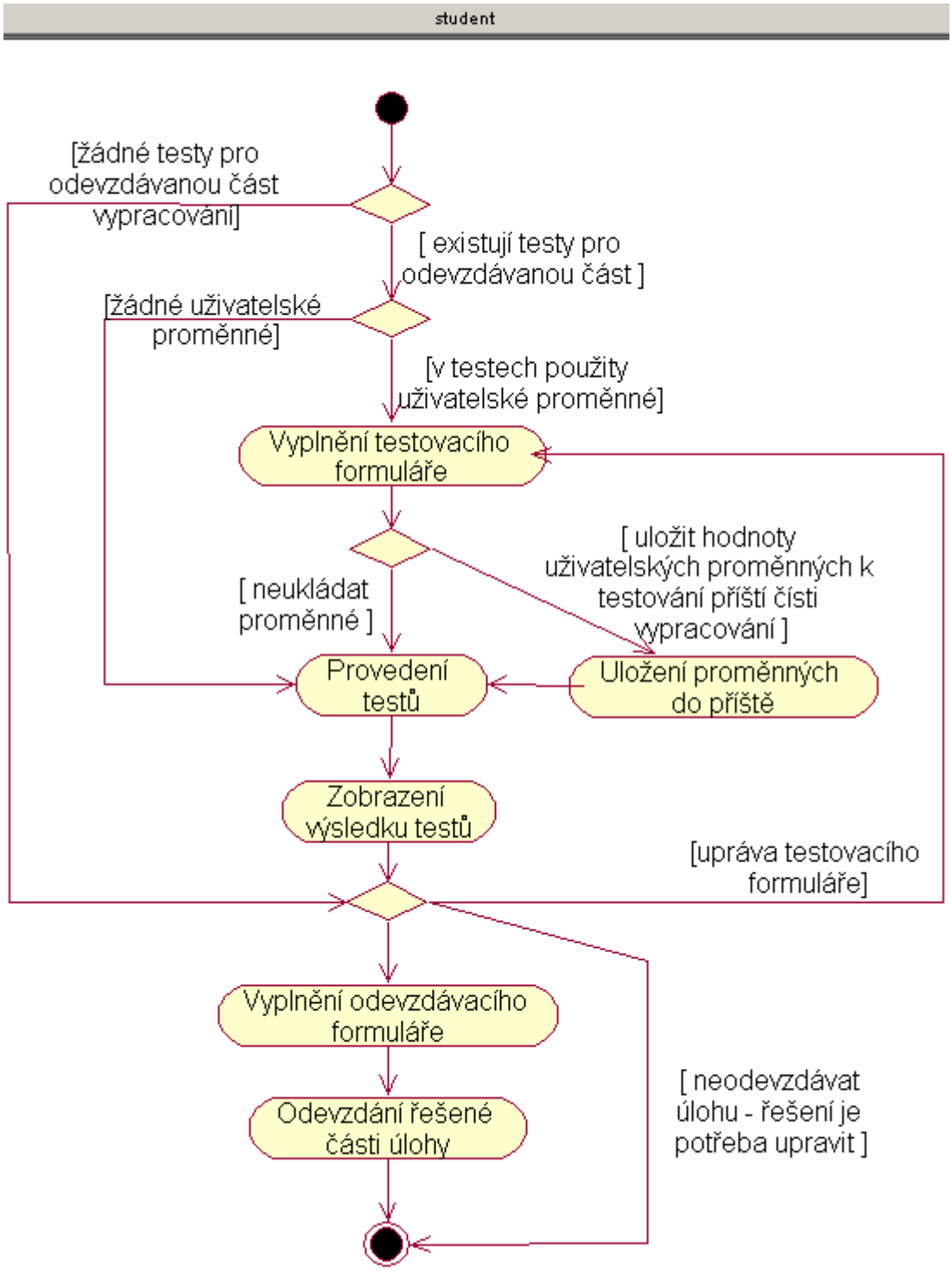


Diagram 2.2.7. Testování a odevzdání části úlohy

2.2.8. Hodnocení řešení

Při zobrazení odevzdaného řešení bude učiteli nabídnut formulář pro hodnocení. Učitel zadá do tohoto formuláře body, které studenti za odevzdané řešení dostanou (na základě výsledků testů a

příloh k řešení) a dále pak učitel volitelně zadá textového hodnocení. Pokud bylo odevzdané řešení pro část vypracování učitelem ohodnoceno, studenti již nemohou znova tuto část vypracovat, i když je část stále otevřena. Znova vyřešit a odevzdat již vyřešenou a ohodnocenou část vypracování bude možné, pokud učitel své hodnocení zruší.

2.2.9. Archivace řešení

Učitel může odevzdaná řešení a hodnocení archivovat, jestliže budou ukončeny všechny části vypracování úlohy. Archivace znamená, že budou řešení uloženy do archivu a úloha bude zrušena (budou odstraněny všechny skupiny vytvořené pro danou šablonu vypracování). Archivování výsledků testů a řešení přináší dvě výhody:

a) Přehlednost – učiteli se na stránce pro správu úloh nezobrazují úlohy, které byly archivovány.
b) Uvolnění prostředků – díky archivaci budou z databáze odstraněna data potřebná na popis skupin zapsaných studentů. Dále pak bude uvolněna šablona vypracování, která může být smazána nebo znovu otevřena (stačí změnit data vypracování jednotlivých části a vytvořit nové skupiny studentů).

Archivované řešení je uloženo v databázové tabulce s agregovanými informacemi v textové podobě.

2.3. Použitá terminologie

Při specifikaci požadavků byly použity termíny, jejichž význam není na první pohled zřejmý. Z tohoto důvodu uvádím seznam použitých termínů s jejich významem (termíny jsou seřazeny podle abecedy). Tyto termíny budou použity i v textu dalších kapitol.

Část vypracování

Jedna úloha může být zadána studentům tak, aby ji vypracovali po částech. Pro každou část vypracování mohou být vybrány testy, které se mají provést nad konfigurovanou topologií a také mohou být specifikovány požadavky na přílohy k řešení. Při definování části vypracování učitel také určí od kdy do kdy mohou studenti danou část vyřešit a odevzdat. Pro každou část řešení může jedna skupina studentů odevzdat jedno řešení.

Odevzdávací formulář

Pokud vyučující definoval požadavky na přílohy k řešení při vytváření části vypracování, studentům bude zobrazen odevzdávací formulář. Do něj studenti vloží požadované informace.

Parametrizace

Slouží k diferenciaci zadání úlohy. Jedna parametrizace představuje jedinečné naplnění parametrizačních proměnných používaných v testech a dále obsahuje slovní popis specifické části zadání úlohy.

Parametrizační proměnná

Hodnotou parametrizační proměnné bude nahrazena proměnná použita ve skriptu testovacího pravidla. Hodnotu parametrizační proměnné zadává učitel při vytváření jednotlivých parametrizací.

Proměnná (ve skriptu testovacího pravidla)

Část skriptu v testovacím pravidle, která bude nahrazena hodnotou uživatelské či parametrizační proměnné.

Řešení

Řešení představuje soubor informací, které studenti odevzdali k jedné části vypracování úlohy. Skládá se z výsledků testů, informací přiložených k řešení (na základě požadavků na přílohy k řešení) a poznámek k řešení.

Skript pravidla

Sekvence příkazů, které budou provedeny na vybraném síťovém prvku. Mezi příkazy lze vkládat pauzy. Pauza pozastaví provádění skriptu na určenou dobu.

Test

Vybrané testovací pravidlo, u kterého učitel určil, jakými parametrizačními či uživatelskými proměnnými budou nahrazeny proměnné ze skriptu pravidla. Dále je specifikováno: a) zařízení, na kterém se má test provést, b) počet bodů, které studenti dostanou za úspěšné splnění, c) povinnost testu – jestli musí být test splněn, aby byl celkový výsledek testování vyhodnocen jako úspěšný

Testovací formulář

Do tohoto formuláře vyplňuje student hodnoty uživatelských proměnných, které jsou použity ve vybraných testech pro studentem řešenou část vypracování.

Testovací pravidlo

Bližší specifikací se z něj vytvářejí testy. Testovací pravidlo určuje, jaké příkazy se mají provést a jak má být rozhodnuto o úspěšnosti/neúspěšnosti splnění testu. Skript pravidla se skládá ze sekvence příkazů, ve kterých lze používat proměnné. Proměnné budou v době provádění testu nahrazeny hodnotami parametrizačních a uživatelských proměnných. Mezi jednotlivé příkazy se vkládají tzv. pauzy, umožňující pozastavit provádění skriptu na určitou dobu. Dále testovací pravidlo obsahuje koncovou značku ukončující načítání konzolového výstupu zařízení, na kterém je test prováděn. A také vyhodnocovací regulární výraz, který bude aplikován na výstup zařízení za účelem rozhodnutí o splnění/nesplnění testu.

Uživatelská proměnná

Hodnotou uživatelské proměnné bude nahrazena proměnná použita ve skriptu testovacího pravidla. Hodnotu uživatelské proměnné zadává student při vyplňování testovacího formuláře.

3. Analýza objektů

V této kapitole hledám a popisuji objekty, které lze identifikovat na základě specifikace požadavků. Popsáním objektů pomocí tříd a upřesněním vazeb mezi objekty vzniká třídní diagram.

V kapitole věnované návrhu (kapitola 4.1. Návrh tříd) se zaměřím na upřesnění tříd z třídního diagramu ve světle implementačního prostředí, uvedu klíčové metody s jejich parametry a návratovými hodnotami.

Stávající řídící aplikace Virtlabu je implementována pomocí jednotlivých dynamických PHP stránek. Každá stránka přistupuje do databáze, provádí část řídící logiky aplikace a výsledek zobrazuje uživateli. Některé složitější funkce řídící logiky jsou vloženy do PHP tříd. Jednou z plánovaných vývojových změn ve Virtuální laboratoři je přepsat implementaci řídící aplikace za použití softwarové architektury Model-View-Controller. Požadavkem na vytvoření mnou navrhovaného projektu tedy je

a) aby jej bylo možné použít ve stávající implementaci řídící aplikace
b) aby jeho případné budoucí zasazení do některého MVC frameworku bylo co nejjednodušší

Zkoumáním problémové domény jsem nalezl níže uvedené objekty, které na konci kapitoly zobrazím v diagramu tříd. Identifikované objekty se dají rozdělit do dvou kategorií:

a) managery – managery představují objekty, které obsahují metody pro provádění řídící logiky systému. Každý manager má minimálně jeden atribut odkazující na datovou vrstvu. Managery také odpovídají za načítání, ukládání a úpravu dat uložených v datové vrstvě.
b) objekty nesoucí data – pro předávání dat mezi managery a PHP stránkami používám jednoduché objekty, které obsahují pouze atributy a k nim přístupové metody. Většina těchto objektů se bude později odrážet v jedné nebo více tabulkách databáze při návrhu E-R diagramu.

3.1. Nalezené objekty

V následujícím přehledu vypisuji informace o jednotlivých managerech a v rámci popisu manageru pak uvádím, se kterými objekty pracuje.

RulesAndVariablesManager

Objekt pro správu testovacích pravidel, testů a proměnných. Pracuje s následujícími objekty:

a) TestingRule – jedno definované testovací pravidlo. Atributy objektu nesou informace, které se u pravidla evidují.
b) Variable - objekt představující uživatelskou či parametrizační proměnnou.
c) Test – informace o jednom testu.

ParametrerizationManager

Manager zodpovědný za administraci parametrizací. Pracuje s objekty Parameterization představující jednotlivé parametrizace.

InstanceManager

Manager obsahující metody pokrývající logiku spravování šablon vypracování, vytváření skupin studentů a jejich párování s parametrizacemi, automatizovaného vytváření skupin studentů, správu odevzdaných řešení a hodnocení a také ukládání hodnot uživatelských proměnných z testovacího formuláře. Pracuje s objekty

a) Instance – přestavuje skupinu studentů s přiřazenou parametrizací a s již odevzdanými řešeními (objekty Result).
b) LineOfParsedFile – při automatizovaném vytváření skupin pomocí souboru bude parserovací metoda generovat pro každý řádek souboru objekt LineOfParsedFile obsahující informace o jednotlivých řádcích.
c) InstanceForm – objekt nesoucí hodnoty uživatelských proměnných vyplněných studenty do testovacího formuláře.
d) Template – šablona vypracování. Objekt nesoucí informace o jedné šabloně vypracování a o jednotlivých částech vypracování (objekty Iteration).

Testing Manager

Objekt provádějící testy. Připojuje se na testovací zařízení, odesílá skripty testovacích pravidel, přijímá výstup a rozhoduje o úspěšnosti/neúspěšnosti testu na základě vyhodnocovacího regulárního výrazu. Pro svou práci používá objekt RulesAndVariablesManager. Výsledky testů a další informace předává okolí prostřednictvím objektů TestResult.

Archive Manager

Zodpovídá za archivaci odevzdaných řešení.

Task Manager

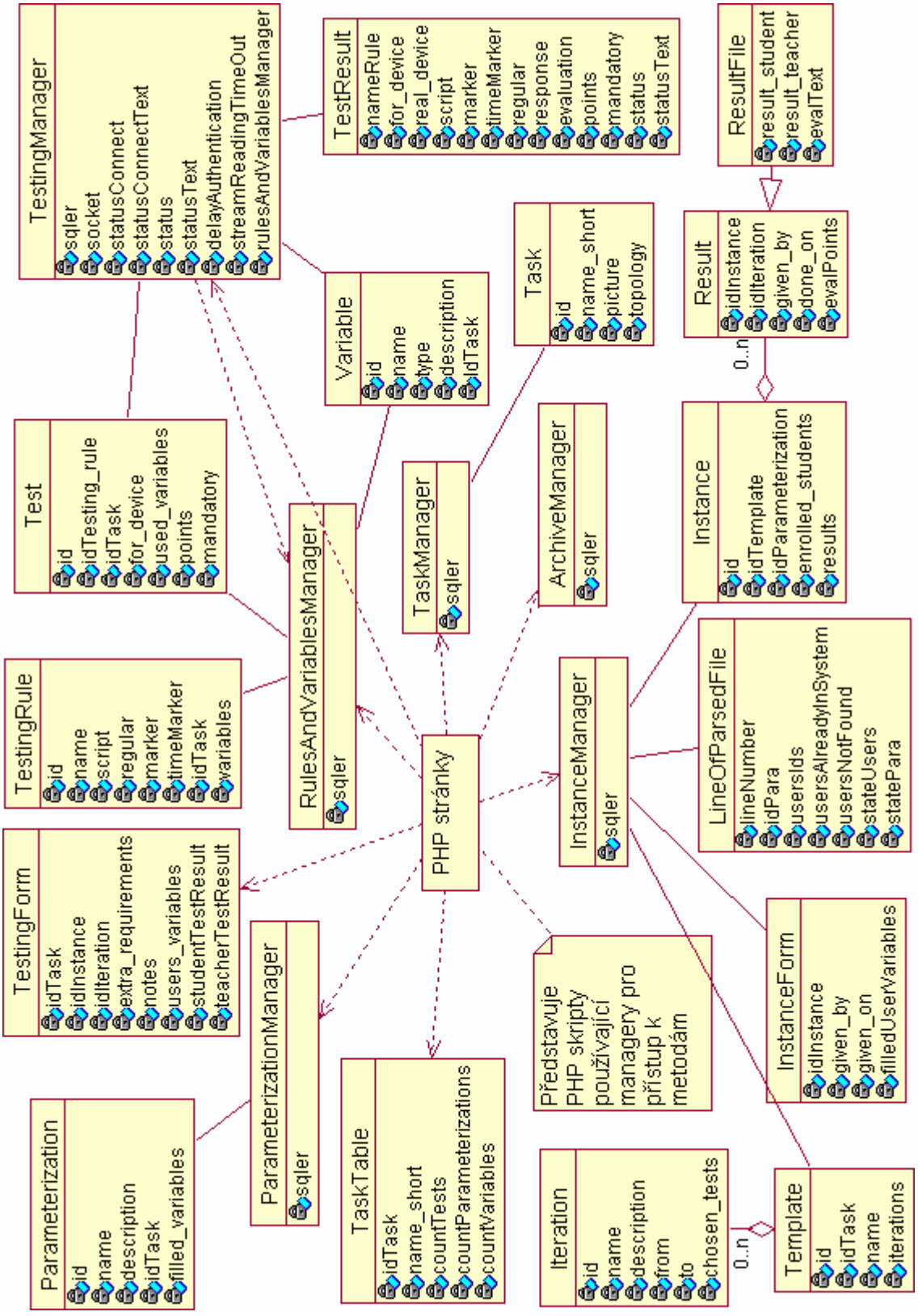
Obsahuje funkce pro načítání informací o úlohách z databáze. Pracuje s objekty Task, které představují jednotlivé úlohy.

TestingForm

Testing form je speciálním objektem, který se bude ukládat do session mezi požadavky relace studenta. Ponese hodnoty uživatelských proměnných vyplněných do testovacího fomruálře a dále pak informace vyplněné studentem do odevzdávacího formuláře. Účelem toho objektu je uchovávat data vyplněné studentem mezi jednotlivými požadavky – aby student nemusel znova vyplňovat veškerá data při návratu na testovací a odevzdávací formulář z jiné stránky.

3.1.1. Diagram tříd

Diagram tříd (obrázek 3.1.1) zobrazuje vztahy nalezených objektů. Uprostřed diagramu se nachází čtverec s popiskem „PHP stránky“, který není třídou, ale představuje soubor PHP stránek, které vytvářejí instance jednotlivých managerů a pomocí nich volají logiku systému.



Obrázek 3.1.1. – Diagram tříd

4. Návrh tříd a databáze

V této kapitole jsou popsány třídy objektů nalezených v kapitole 3. Analýza objektů a navržen E-R diagram pro datovou vrstvu.

4.1. Návrh tříd

Jednotlivé třídy upřesním ve světle implementačního prostředí, uvedu datové typy atributů a hlavně pak metody s parametry a návratovými hodnotami. Systém automatizace hodnocení konfigurací bude implementován v programovacím jazyce PHP, proto navrhované datové typy proměnných budou odpovídat datovým typům jazyka PHP.

U jednotlivých metod a atributů uvádím pouze základní informace. Podrobnější informace lze nalézt v komentářích, které se nacházejí ve zdrojových kódech.

4.1.1. Upřesnění tříd objektů nesoucích data

Všechny třídy, které popisují objekty sloužící pro předávání informací mezi stránkami a řídící logikou aplikace, se skládají z privátních atributů a z veřejných metod pro jejich získávání (předpona set) a nastavování (předpona get). U obrázků tříd reprezentujících tyto objekty jsou uvedeny vždy pouze dvě metody set() a get(), které zastupují získávací a nastavovací metody každého atributu ve tvaru getJmenoAtributu a metodu setJmenoAtributu.

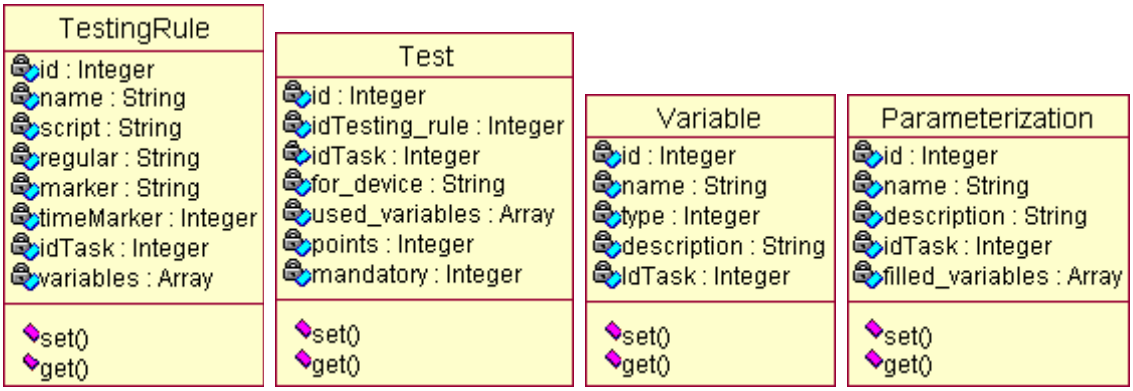
V některých třídách se nachází atribut Id, který jednoznačně identifikuje objekt v množině objektů instanciovaných ze stejné třídy. Tyto objekty budou načítány/ukládány do databáze, kde v tabulce určené pro uchovávání daného typu objektu bude atribut id primárním klíčem.

Třída testovacího pravidla (obrázek 3.2.1.a – TestingRule) obsahuje atributy představující identifikátor pravidla (id), jméno (name), skript, vyhodnocovací regulární výraz (regular), koncovou značku (marker), časovou koncovou značku (timeMarker), identifikátor úlohy, pro kterou bylo pravidlo vytvořeno (idTask) a pole proměnných (variables), které pravidlo používá. Pole variables je asociativní pole, kde klíčem je jméno (kód) proměnné použité ve skriptu a hodnotou je uživatelsky příjemný název proměnné.

Atributy testu (obrázek 3.2.1.a – Test) znamenají identifikátor testu, identifikátor testovacího pravidla (idTesting_rule), ze kterého je vytvořen, identifikátor úlohy, zařízení na kterém se má test provést (for_device), body za test (points) a jestli je test povinný (mandatory). Atribut used_variables je pole nesoucí informace o přiřazení uživatelských či parametrizačních proměnných (objekt Variable) proměnným obsažených ve skriptu pravidla, ke kterému byl daný test vytvořen. Asociativní pole má tvar – klíč: jméno proměnné použité ve skriptu a hodnota: identifikátor vybrané proměnné (objekt Variable).

Třída pro popis objektů proměnná (obrázek 3.2.1.a – Variable) se skládá z identifikátoru proměnné, jména, druhu (nabývá hodnot 1: parametrizační proměnná, 2: uživatelská proměnná – atribut type), popisu a identifikátoru úlohy (idTask), ke které proměnná patří.

Parametrizace (obrázek 3.2.1.a – Parameterization) obsahuje identifikátor parametrizace, jméno, popisek, identifikátor úlohy a seznam vyplněných parametrizačních proměnných, které učitel vyplnil pro danou parametrizaci (filled_variables).

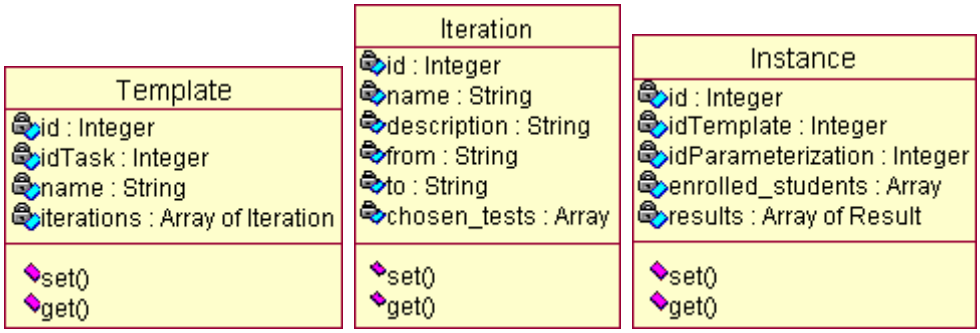


Obrázek 3.2.1.a – Třídy TestingRule, Test, Variabla a Parameterization

U šablony vypracování (obrázek 3.2.1.b – Template) nás zajímá identifikátor šablony, identifikátor úlohy, pro kterou byla šablona vytvořena, jméno a nakonec seznam jednotlivých částí vypracování (pole iterations obsahující objekty Iteration)

Část vypracování (obrázek 3.2.1.b – Iteration) nese atributy znamenající identifikátor části vypracování, jméno, popisek, datum a čas začátku (from) a konce (end) vypracování a seznam identifikátorů vybraných testů (atribut chosen_rules), které mají být aplikovány na topologii při odevzdávání dané části.

Skupinu studentů s přiřazenou parametrizací reprezentuje objekt Instance (obrázek 3.2.1.b – Instance). Obsahuje identifikátor skupiny studentů, identifikátor šablony vypracování, identifikátor vybrané parametrizace, seznam studentů patřících do skupiny a také seznam odevzdaných řešení. Seznam studentů je pole (enrolled_students) ve formátu – klíč: identifikátor uživatele a hodnota: jméno uživatele. Seznam řešení je uložen v poli results a obsahuje objekty typu Result.

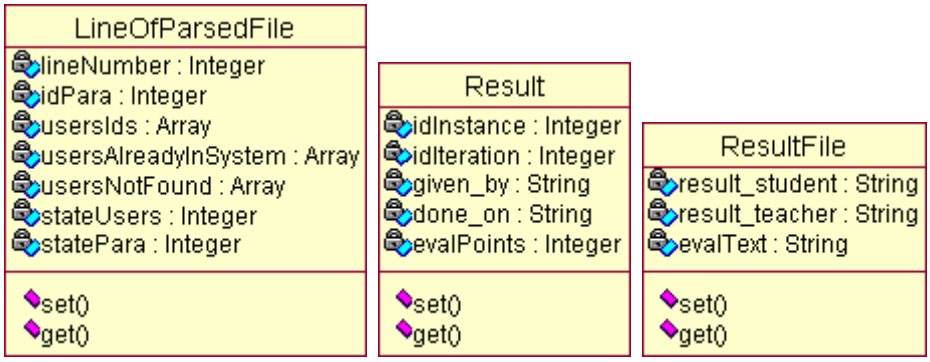


Obrázek 3.2.1.b – Třídy Template, Iteration a Instance

Při automatizovaném vytváření skupin studentů generuje parserovací metoda pro každý řádek souboru objekt LineOfParsedFile (obrázek 3.2.1.c). Tento objekt obsahuje informace o čísle řádku v souboru, identifikátoru parametrizace (idPara), identifikátorech studentů, kteří byli nalezeni na řádku (usersIds), identifikátorech studentů, kteří byli nalezeni, ale již jsou v nějaké skupině pro danou šablonu vypracování (usersAlreadyInSystem), identifikátorech studentů, které neexistují v databázi řídící aplikace (usersNotFound). Dále objekt nese číselné hodnoty vyjadřující stav nalezených uživatelů (stateUsers) a parametrizací (statePara). Stavy znamenají, jestli byl řádek rozpoznán v pořádku nebo nastala chyba. Hodnoty, kterých můžou nabývat atribut stateUsers a statePara lze nalézt v komentářích zdrojového kódu.

Objekt Result (obrázek 3.2.1.c) představuje základní informace o odevzdaném řešení studenty k jedné části vypracování. Skládá se z identifikátoru skupiny studentů (idInstance), která řešení odevzdala, identifikátoru části vypracování, ke které bylo odevzdáno, identifikátoru studenta (given_by), který řešení za skupinu odevzdal, datum a čas odevzdání (done_on) a v případě, že bylo řešení ohodnoceno vyučujícím, také získané body (evalPoints).

Objekt ResultFile (obrázek 3.2.1.c) vzniká rozšířením objektu Result a obsahuje navíc výsledky testů a informace vyplněné do odevzdávacího formuláře v textové podobě (result_student), podrobnou zprávu testování v textové podobě (result_teacher) a nakonec pak textové hodnocení, které vložil učitel k tomuto řešení. Objekt ResultFile představuje kompletní informace o výsledcích testů, odevzdaném řešení a hodnocení. Naproti němu existuje objekt Result, který vznikl z důvodů optimalizace. Je používán výhradně objektem Instance a obsahuje jen základní informace o odevzdaném řešení. Při práci se systémem se v různých částech budou zobrazovat informace jestli skupina odevzdala/neodevzdala řešení apod., ale nebude potřeba zobrazovat obsah řešení a hodnocení.

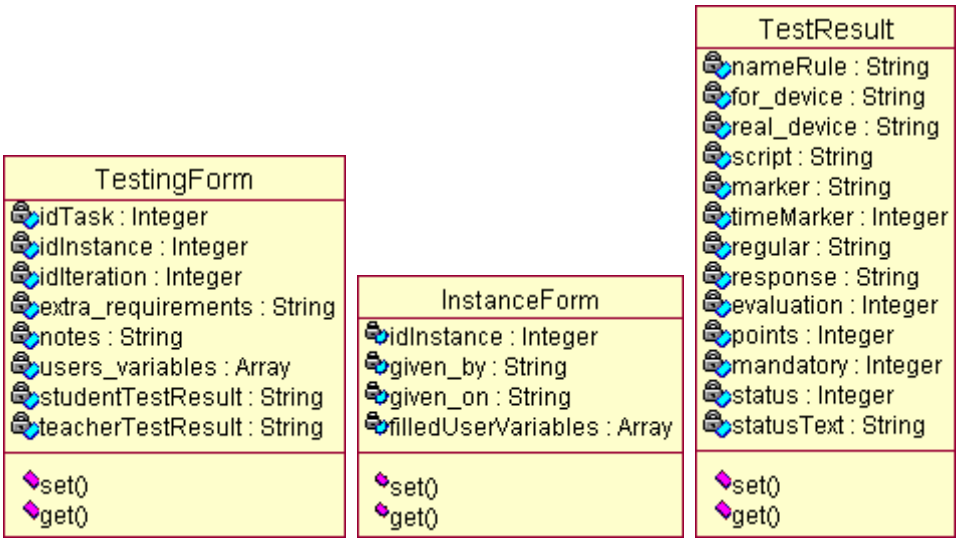


Obrázek 3.2.1.c – Třídy LineOfParsedFile, Result, ResultFile

Testovací formulář (obrázek 3.2.1.d – TestingForm) bude ukládán do session mezi požadavky uživatele a ponese informace vyplněné do testovacího a odevzdávacího formuláře a také textovou podobu výsledku testů. Atributy objektu jsou: identifikátor úlohy, pro kterou se úloha odevzdává, identifikátor skupiny studentů (idInstance), identifikátor části vypracování (idIteration), přílohy k řešení (extra_requierements), poznámky k řešení (notes), hodnoty vyplněných uživatelských proměnných (users_variables) a v případě, že již student provedl testování také agregovaná textová zpráva výsledku testů (studentTestResult) a podrobnou zpráva testování (teacherTestResult).

Objekt InstanceForm (obrázek 3.2.1.d) nese hodnoty uživatelských proměnných, které byly uloženy některým studentem ze skupiny. Jestliže kdokoli z řešící skupiny studentů uložil hodnoty uživatelských proměnných do příště, jsou ostatním tyto hodnoty zobrazeny při vyplňování testovacího formuláře.

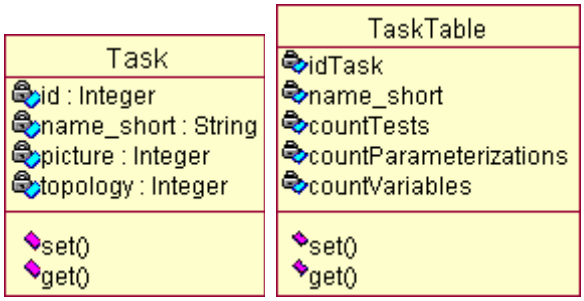
Objekty TestResult (obrázek 3.2.1.d) generuje TestingManager při provádění jednotlivých testů. Každá instance třídy TestResult obsahuje informace o jednom provedeném popř. neprovedeném testu. Atributy znamenají jméno prováděného pravidla (nameRule), jméno prvku v logické topologii (for_device), jméno – identifikátor fyzického prvku (real_device), prováděný skript, koncová značka (marker), časová koncová značka (timeMarker), vyhodnocovací regulární výraz na ověření úspěšnosti provedení testu (regular), odpověď fyzického prvku po odeslání skriptu (response), vyhodnocení úspěšnosti testu po aplikování regulárního výrazu na odpověď prvku (evaluation), získané body za test vyhodnocený jako úspěšně provedený (points), povinnost testu (mandatory) a nakonec číselná hodnota vyjadřující stav provedení testu (status) a textová zpráva blíže specifikující stav provedení (statusText). Atribut status může nabývat tří hodnot: 1 – test proběhl v pořádku, 2 – nepodařilo se připojit k testovanému zařízení, 3 – nastala chyba při provádění testu.



Obrázek 3.2.1.d – třídy TestingForm, InstanceForm, TestResult

Třída Task (obrázek 3.2.1.e) popisuje úlohu Virtlabu. Obsahuje pouze atributy potřebné z hlediska navrhovaného systému. Jsou jimi identifikátor úlohy, jméno úlohy (name_short), identifikátor souboru zobrazujícího topologii úlohy (picture) a nakonec identifikátor souboru obsahujícího popis logické topologie úlohy (topology). Soubory obrázek a popis topologii jsou uloženy v databázi v tabulce files pod číselným identifikátorem.

Objekt TaskTable (obráek 3.2.1.e) slouží k zobrazování seznamu úloh, ve kterém je uveden počet testů, parametrizací a proměnných vytvořených pro každou úlohu. Jeden objekt nese informace o jedné úloze. Atributy této třídy jsou: identifikátor úlohy, jméno úlohy, počet testů definovaných testů (countTests) a proměnných (countVariables) pro tuto úlohu a počet vytvořených parametrizací (countParameterizations).



Obrázek 3.2.1.e – třídy Task a TaskTable

4.1.2. Upřesnění tříd managerů

Objekty managerů tvoří řídící logiku systému. Přistupují do databáze, načítají a ukládají objekty pro přenos dat (viz. kap. 3.2.1.) a obsahují další metody potřebné pro chod systému. Každý manager má minimálně jeden atribut zvaný sqler. Při vytváření instance managera je v jeho konstruktoru přiřazen atributu sqler odkaz na instanci třídy virtlabSQL představující rozhranní pro přístup do databáze. Třída virtlabSQL je již implementována v řídící aplikaci.

V následujícím přehledu managerů vyjádřím smysl jednotlivých managerů a popíši některé zajímavé metody. Podrobný popis metod se nachází v komentářích zdrojových kódů, které jsou přiloženy na disku CD.

Metody managerů se snažím pojmenovávat co nejvýstižněji za účelem snadného určení jejich významu.

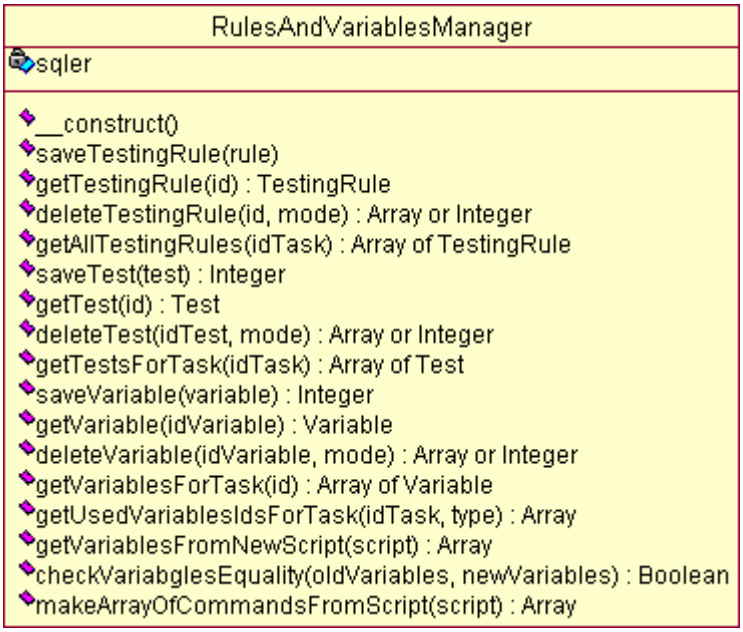
RulesAndVariablesManager

RulesAndVariablesManager (obrázek 3.2.2.a) zodpovídá za agendu spojenou s testovacími pravidly, testy a proměnnými. Obsahuje metody pro načítání a ukládání objektů TestingRule, Test a Variable z/do databáze. A také pro mazání jejich reprezentací z databáze.

Při mazání se kromě parametru identifikujícího mazaný záznam předává taky parametr určující mód mazání. Testovací pravidlo nemůže být smazáno, pokud bylo použito k vytvoření jakéhokoli testu a dále pak proměnná nemůže být smazána, když je použita pro definici jakéhokoli testu. Test nemůže být smazán, jestliže je vybrán k provedení v jakékoli části vypracování . V případě, že mazací funkce zjistí, že záznam nemůže být smazán z důvodů existence závislosti a zároveň je parametr mode nastaven na hodnotu 1, vrátí mazací funkce seznam částí vypracování, respektive seznam testů, na kterých je mazaný záznam závislý. V opačném případě vrací funkce celočíselný stavový kód. Kód může nabýt hodnot: 0 – smazáno, -1 – nesmazáno z důvodu databázové chyby, -2 – nesmazáno z důvodu existence závislosti.

Při vytváření testovacích pravidel se uplatní funkce *getVariablesFromNewScript*, která v zadaném skriptu pravidla hledá případné proměnné. Pro potřeby přehledného výpisu skriptu nebo pro testování se používá metoda *makeArrrayOfCommandsFromScript*, která vrací pole s jednotlivými řádky (příkazy) či pauzami nalezenými ve skriptu pravidla.

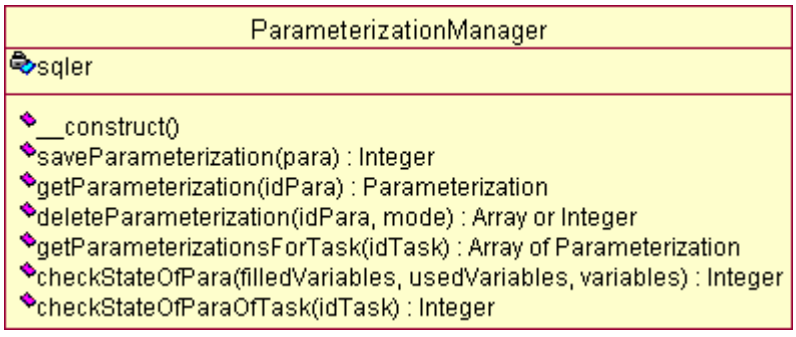
Dále manager obsahuje metody pro získávání kolekcí objektů podle různých kritérií. Například metoda *getTestsForTask* vrací pole objektů Test pro úlohu identifikovanou parametrem idTask.



Obrázek 3.2.2.a – RulesAndVariablesManager

ParameterizationManager

Parameterization Manager (obrázek 3.2.2.b) spravuje jednotlivé parametrizace. Umožňuje je ukládat, načítat a mazat. Zajímavou funkcí je *checkStateOfPara* kontrolující, jestli jsou vyplněny hodnoty všech parametrizačních proměnných, které byly použity pro definici testů. Obdobu této funkce představuje funkce *checkStateofParaOfTask* testující, jestli jsou všechny parametrizace vytvořené pro danou úlohu v pořádku. V pořádku znamená, že jsou ve vytvořených parametrizacích vyplněny hodnoty všech parametrizačních proměnných, které mají být použity v testech pro danou úlohu.



Obrázek 3.2.2.b – ParameterizationManager

InstanceManager

Instance manager (obrázek 3.2.2.c) má mimo jiné na starost správu (ukládání, načítání a mazání) šablon vypracování, jednotlivých část vypracování, skupin studentů, odevzdaných řešení a uživatelských proměnných.

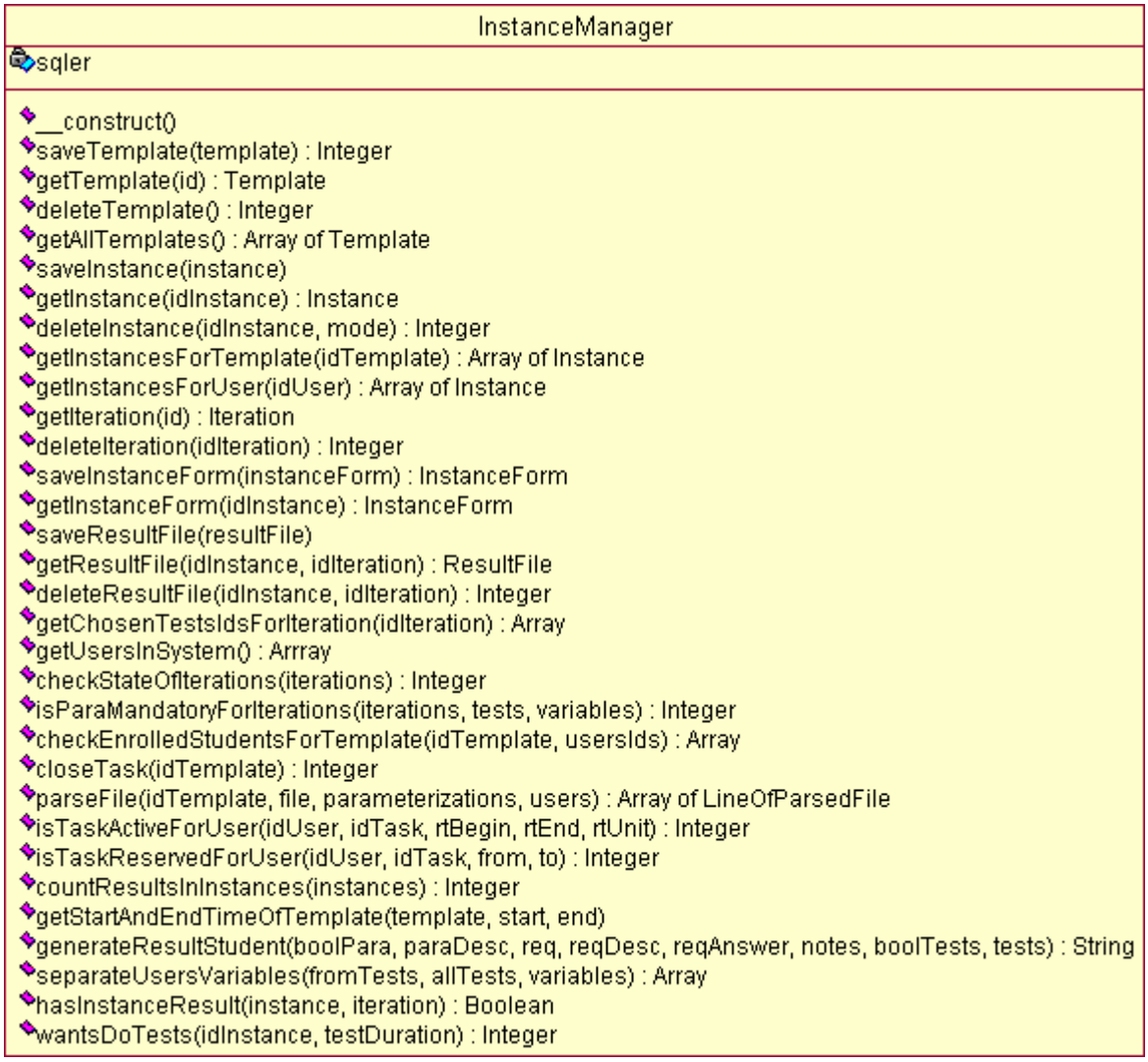
Při vytváření skupin studentů se uplatňuje metoda *checkEnrolledStudentsForTemplate*, která ověří, jestli studenti, kteří mají být přiřazeni do nové skupiny, již nejsou přiřazeni do skupiny jiné. Při automatizovaném vytváření studentů pomocí souboru se využívá metody *parseFile*, která projde vložený soubor a pro každý řádek souboru vrací analyzovaný výsledek prostřednictvím objektu LineOfParsedFile. Metoda *isParaMandatoryForIterations* zjišťuje, jestli by učitel měl při vytváření skupin studentů přiřadit skupině parametrizaci nebo ne. Pokud je k jakékoli části vypracování vybrán test, který používá parametrizační proměnnou, měla by být vybrána parametrizace.

Pro zjištění, jestli je daná úloha rezervována či dokonce aktivní pro přihlášeného uživatele slouží metody *isTaskActiveForUser* a *isTaskReservedForUser*.

Vypracovávat a řešit úlohu mohou všichni studenti ze skupiny. Taktéž mohou provádět testování. Z důvodů správného provedení testování je nežádoucí, aby se v jednu chvíli provádělo více testování na jedné topologii. Aby bylo zaručeno, že v rámci jedné rezervované úlohy probíhá pouze jedno testování, volá se před začátkem každého testování metoda *wantsDoTests*. Tato metoda ověří, jestli již pro danou skupinu studentů není spuštěno testování nad vypracovávanou úlohou a v případě, že ano, vrátí čas, kdy bude možné provést další testování. V případě, že není prováděno testování, metoda zajistí vyhrazení potřebného času pro testování na základě délky testování (parametr testDuration) a vrátí hodnotu 0 indikující úspěšnost vyhrazení času. Vyhrazení času znamená, že je vložen do databázové tabulky záznam indikující provádění testování ve vyhrazeném čase.

Pro informování studentů a učitelů o stavu odevzdaných řešení slouží metody *countResultsInInstances*, která zjišťuje celkový počet odevzdaných řešení k otevřené úloze (parametrem je kolekce objektů Instance přiřazených k šabloně vypracování) a dále metoda *hasInstanceResut*, která zjišťuje, jestli skupina studentů odevzdala řešení pro vybranou část řešení.

Při odevzdávání řešení studentem bude volána metoda *generateResultStudent*, která z informací vyplněných do odevzdávacího formuláře a z výsledkové zprávy testování vygeneruje jednoduchý agregovaný html dokument, který bude uložen do databáze.



Obrázek 3.2.2.c – InstanceManager

TestingManager

Testing manager (obrázek 3.2.2.d) byl navrhnut za účelem provádění testování. Připojuje se na prvky topologie, odesílá jim příkazy definované ve skriptu testovacího pravidla, získává odpověď a vyhodnocuje úspěšnost a neúspěšnost testu.

Základní metodou testovacího manageru je metoda *test*, která prostřednictvím parametrů přijímá část informací potřebných pro provedení testů. Jsou jimi testy, které mají být provedeny, a hodnoty parametrizačních a uživatelských proměnných, kterými mají být nahrazeny proměnné použité ve skriptu pravidla. Další informace potřebná pro provedení testů je mapování logických prvků topologie na prvky fyzické. Informace o namapování logických prvků na fyzické pro aktuálně rezervovanou úlohu lze zjistit prostřednictvím funkce *getParsedDevices*, která tyto údaje vyčte s databáze.

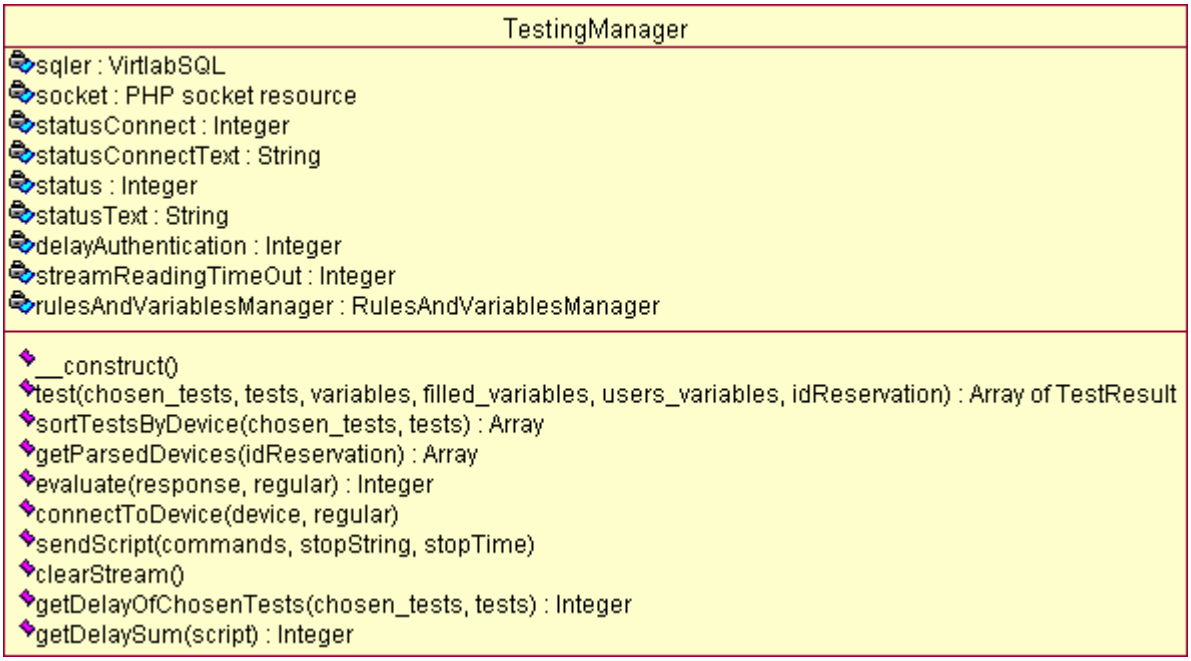
Metoda *test* si nejprve seřadí testy k provedení podle zařízení, na kterém mají být provedeny, pomocí funkce *sortTestsByDevice*. Následně metoda prochází seřazené testy podle zařízení a pomocí

funkce *connectToDevice* se připojuje na fyzické prvky topologie prostřednictvím konzolového serveru. Funkce *connectToDevice* vytváří síťový socket a odkaz na něj ukládá do privátní proměnné *socket*. Odesílání skriptu (jednotlivých příkazů) fyzickému zařízení provádí funkce *sendScript*. Ta nejprve vyčistí přijímací buffer otevřeného socketu voláním funkce *clearStream* pro případ, že se v bufferu objevil případný výstup z pravidla minulého. Poté do socketu zapisuje jednotlivé příkazy skriptu. V případě, že metoda narazí na pauzu, pozastaví zapisování příkazů skriptu do socketu na délku pauzy. Metoda sendScript po odeslání všech příkazů vrací výstup, který načetla ze socketu po odeslání všech příkazů. Metoda *test* pak pomocí funkce *evaluate* vyhodnotí, jestli získaný výstup odpovídá vyhodnocovacímu regulární výrazu (test vyhodnocen jako úspěšný). Pro každý provedený test vygeneruje metoda *test* objekt TestResult obsahující informace o právě provedeném testu. Po skončení testování (provedení všech testů) vrací metoda *test* pole obsahující všechny vytvořené objekty TestResult.

Při vytváření instance TestingManager jsou v konstruktoru nastaveny privátní proměnné delayAuthentication a streamReadingTimeOut. Hodnoty, které mají být nastaveny, vyčte konstruktor ze souboru settings.php obsahující nastavení řídící aplikace. DelayAuthentication určuje, jak dlouho se má čekat, než se začne načítat odpověď z konzolového serveru po odeslání autentizačních údajů. Konzolový server v nové implementaci po navázání spojení odesílá informace o úspěšnosti/neúspěšnosti zprostředkování přístupu na dané zařízení po dvou vteřinách po odeslání autentizačních údajů. StreamReadingTimeOut určuje, jak dlouho se má počkat při načítání dat ze socketu, pokud nastane „hladovění“ – pokud v bufferu socketu nejsou žádná data ke čtení.

Privátní proměnné statusConnect a statusConnectText nesou stavové kódy a případná chybová hlášení, která jsou generována při připojování k zařízení. Jestliže nastane chyba v metodě *connectToDevice* při vytváření socketu nebo při autentizaci ke konzolovému serveru, je proměnná statusConnect nastavena na hodnotu určující typ chyby a statusConnectText je naplněna chybovým hlášením. Proměnná statusConnect může nabývat těchto hodnot: 1 – úspěšně připojené, 2 – nastala chyba při připojování.

Podobný účel jako proměnné statusConnect a statusConnectText pak mají také proměnné status a statusText, které jsou nastavovány v metodě *sendScript* při odesílání skriptu zařízení či načítání odpovědi. Status může nabývat následujících hodnot: 1 – v pořádku provedeno, 2 – nastala chyba.



Obrázek 3.2.2.d – TestingManager

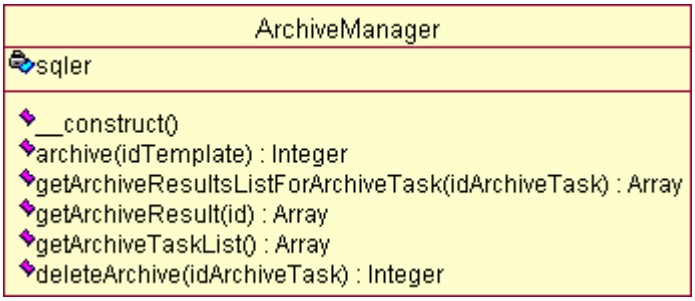
ArchiveManager

ArchiveManager (obrázek 3.2.2.e) slouží k archivaci odevzdaných řešení a k provádění úkonů s archivací spojených. Samotný proces archivace provádí metoda *archive*. Tato metoda z datového úložiště odstraní informace o vytvořených skupinách studentů, které byly vytvořeny k šabloně vypracování. Odevzdaná řešení a základní informace o archivované úloze (šabloně vypracování) uloží do zvláštních databázových tabulek, které nebudou mít vazbu na žádné jiné tabulky.

Procesem archivace se „uvolní“ šablona vypracování, ke které byly smazané skupiny studentů přiřazeny. Uvolnění znamená, že šablona může být smazána nebo znovu otevřena. Pokud se vyučující rozhodne šablonu vypracování znovu použít, bude postačující, když pouze upraví začátky a konce vypracování jednotlivých částí vypracování dané šablony. Specifikace požadavků na přílohy a vybraná testovací pravidla, která se u jednotlivých části mají provést, zůstávají u šablony zachovány. V případě odstranění šablony vypracování bude také možné odstranit testy, které byly vybrány v některé části vypracování smazané šablony (pokud nejsou přiřazeny k částem vypracování jiné šablony pro stejnou úlohu). Smazáním testů je pak možné odstranit jednotlivá testovací pravidla, která byla použita pro definici smazaných testů.

Pro potřeby zobrazování archivovaných úloh s archivovanými řešeními slouží metody *getArchiveTaskList* (vrací seznam archivovaných úloh), *getArchiveResultsListForArchiveTask* (vrací seznam archivovaných řešení pro vybranou úlohu) a *getArchiveResult* (vrací vybrané řešení)

Metoda *deleteArchive* odstraní archivovanou úlohu se všemi archivovanými řešeními.

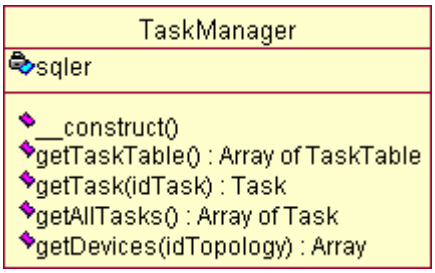


Obrázek 3.2.2.e – ArchiveManager

TaskManager

TaskManager (obrázek 3.2.2.f) získává informace o jedné nebo více úlohách existujících v databázi lokality (metody *getTask*, *getAlTasks* a *getTaskTable*).

Funkce getDevices vrací seznam logických názvů prvků použitých v definici topologie úlohy.



Obrázek 3.2.2.f – TaskManager

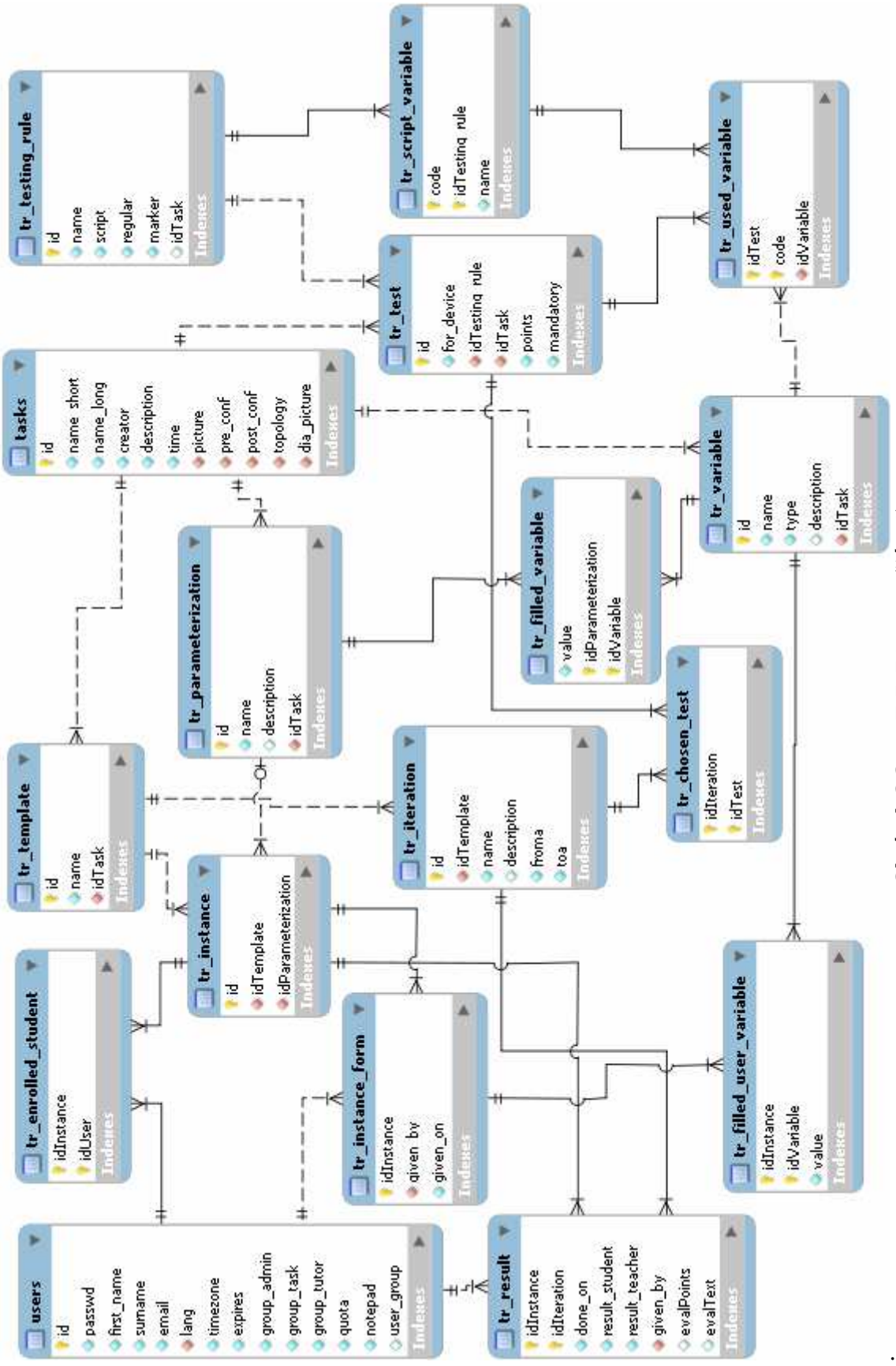
V této kapitole se mi podařilo navrhnout třídy objektů obsahujících řídící logiku vytvářeného systému (managery) a třídy objektů pro přenos dat (datové objekty). V implementační fázi bude vytvořeno grafické uživatelské rozhraní (GUI) pomocí dynamických PHP stránek. Tyto stránky budou přistupovat k řídící logice prostřednictvím volání metod jednotlivých managerů. Data mezi řídící logikou a stránkami se předávají prostřednictvím objektů pro přenos dat.

V případě transformování řídící aplikace na architekturu MVC by jednotlivé PHP stránky vytvářely komponentu view a navržené objekty managerů a datových objektů, by představovaly model aplikace. Controller, který by reagoval na události generované uživateli a volal metody modelu aplikace, by musel být doimplementován.

4.2. Návrh databáze

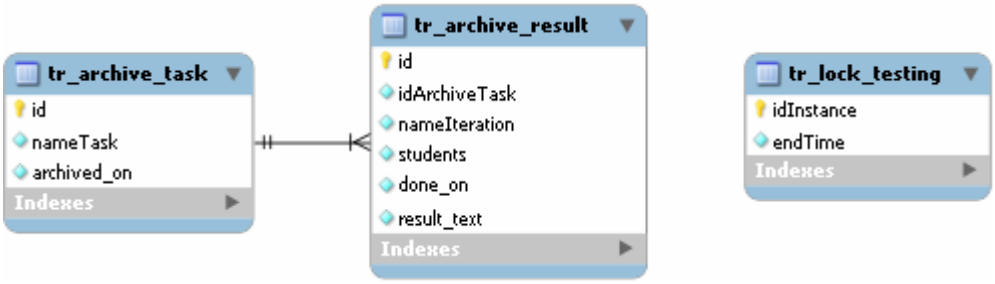
Řídící aplikace Virtuální laboratoře používá jako datovou vrstvu pro ukládání informací databázi MySQL [4][5] s typem tabulek InnoDB. Na základě navržených tříd a specifikovaných požadavků jsem navrhnul strukturu relační databáze, kterou zobrazují obrázky 3.3.a E-R diagram (část

1) a 3.3.b E-R diagram (část 2). Význam jednotlivých databázových tabulek shrnuje tabulka 3.3. Datový slovník a integritní omezení jsou uvedeny v příloze A – Datový slovník a integritní omezení.



Obrázek 3.3.a – E-R Diagram (část 1)

.



Obrázek 3.3.b – E-R diagram (část 2)

V databázi Virtuální laboratoře již existují tabulky Tasks (obsahuje úlohy Virtlabu) a Users (uživatelé Virtlabu). Zbytek tabulek z obrázku 3.3.a a 3.3.b budou přidány. Tabulka 3.3. obsahuje stručný popis navržených tabulek. Ve sloupci „Odpovídá objektu či atributu“ je uveden název datového objektu (objekty definované v kapitole 3.2.1) nebo atributu s názvem patřičného objektu, které reprezentují danou tabulku v systému.

Tabulka	Popis tabulky	Odpovídá objektu či atributu
Tasks	Tabulka pro ukládání úloh Virtuální laboratoře, (tato tabulka již existuje v databázi)	Objekt Task
Users	Obsahuje seznam uživatelů (tabulka již existuje v databázi)	
Tr_testing_rule	Obsahuje informace o testovacích pravidlech, slouží pro ukládání objektů TestingRule.	Objekt TestingRule
Tr_script_variable	Proměnná použitá ve skriptu testovacího pravidla, slouží pro ukládání atributu variables objektu TestingRule.	Atribut variables objektu TestingRule
Tr_test	Slouží pro ukládání informací o testech.	Objekt Test
Tr_used_variable	Tabulka nese informace o proměnných, které byly použity pro definici testu (jakými proměnnými mají být nahrazeny proměnné použité ve skriptu testovacího pravidla).	Atribut used_variables objektu Test
Tr_variable	Obsahuje seznam proměnných pro jednotlivé úlohy.	Objekt Variable
Tr_parameterization	Tabulka pro uchovávání informací o vytvořených parametrizacích.	Objekt Parameterization
Tr_filled_variable	Hodnoty vyplněných parametrizačních proměnných. Vazební tabulka mezi tr_parameterization a tr_variable.	Atribut filled_variables objektu Parameterization
Tr_template	Obsahuje seznam šablon vypracování	Objekt Template
Tr_iteration	Části vypracování	Objekt Iteration

Tr_chosen_test	Vazební tabulka mezi tr_iteration a tr_test. Představuje identifikátory testů, které se mají provést při odevzdávání části vypracování.	Atribut chosen_tests objektu Iteration
Tr_instance	Představuje skupinu studentů s přiřazenou parametrizací. Zapsaní studenti jsou uloženi v tabulce tr_enrolled_student.	Objekt Instance
Tr_enrolled_student	Obsahuje identifikátory studentů zapsaných do daných skupin studentů. Vazební tabulka mezi tr_instance a users	Atribut enrolled_students objektu Instance
Tr_instance_form	Obsahuje informace kým a kdy byly uloženy hodnoty uživatelských proměnných do dalšího řešení úlohy.	Objekt InstanceForm
Tr_filled_variables	Vazební tabulka mezi tr_instance_form a tr_variables s uloženou hodnotou uživatelské proměnné.	Atribut filledUserVariables objektu InstanceForm
Tr_result	Odevzdané řešení skupinou studentů k části vypracování. Obsahuje také případné hodnocení pedagogem.	Objekt ResultFile nebo Result
Tr_archive_task	Obsahuje informace o archivované úloze	
Tr_archive_result	Obsahuje agregované informace o jednotlivých archivovaných řešeních.	
Tr_lock_testing	Slouží k „zamykání“ testování. Zajišťuje, aby v jednu chvíli probíhalo maximálně jedno testování pro každou skupinu studentů pro každou část vypracování.	

Tabulka 3.2 – Popis navržených databázových tabulek

5. Implementace

Implementační část spočívala ve vytvoření navržených tříd, naprogramování patřičných metod a vytvoření uživatelského rozhraní pro práci se systémem. Implementace tříd a metod probíhala na základně kapitoly 4. Návrh, která velmi podrobně popisuje jednotlivé třídy.

Uživatelskému rozhraní se blíže věnuje kapitola 5.1. Uživatelské rozhraní.

5.1. Uživatelské rozhraní

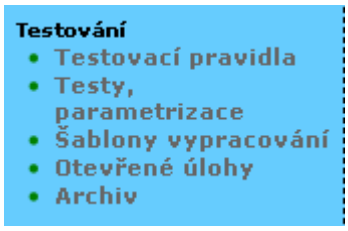
Uživatelské rozhraní je realizováno pomocí dynamických PHP stránek [5][6], které jsou zasazeny do řídící aplikace. Způsob vkládání nových stránek do řídící aplikace popisuje kapitola 1.3.1. Stránky generují XHTML kód. Pro definici vzhledu uživatelského rozhraní používám technologie CSS. Grafické rozvržení a vzhled stávajících stránek je definován v souboru virtlab-style.css. Za účelem jednotného vzhledu uživatelského rozhraní vytvářeného systému a řídící aplikace používám také kaskádových stylů definovaných ve zmíněném souboru.

Pro realizace stránek uživatelského rozhraní byly použity výše zmíněné technologie PHP, XHMTL a CSS. Na stránce pro definici testů byla navíc použita technologie AJAX a blíže se této stránce věnuje kapitola 5.1.2. Pro usnadnění zadávání data a času pro začátek a konec částí vypracování jsem využil externí komponenty pro výběr data a času (viz. kapitola 5.1.3.)

Každá PHP stránka slouží k zobrazování informací a také zodpovídá za zpracování událostí. Pokud uživatel vygeneruje událost (např. odešle formulář, klikne na ikonu pro smazání testu), PHP stránka, na kterou je požadavek s událostí odeslán, vytvoří instanci odpovídajícího managera a zavolá metodu či více metod, pomocí kterých danou událost obslouží. Výsledek pak zobrazí uživateli.

5.1.1. Zasazení stránek do řídící aplikace

Hlavní menu řídící aplikace bylo pro učitele (v současné implementaci uživatelé s právy administrátor a task manager) rozšířeno o submenu Testování s položkami, které zobrazuje obrázek 5.1.1.a.

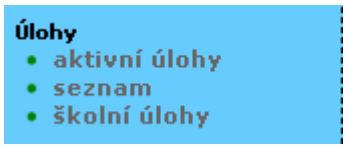


Obrázek 5.1.1.a – rozšíření menu (učitel)

Pod odkazem Testovací pravidla se skrývá soubor stránek umožňující učiteli vytvářet, editovat a mazat testovací pravidla. Položka menu „Testy, parametrizace“ odkazuje na stránky, pomocí kterých

lze spravovat jednotlivé testy a také parametrizace. Odkaz Šablony vypracování slouží ke zobrazení uživatelského rozhraní pro vytváření a správu šablon vypracování a také jejich částí. Pod Otevřenými úlohami se skrývá seznam šablon vypracování, ke kterým byly vytvořeny skupiny studentů s případně přidělenou parametrizací. Pokud je k šabloně vypracování vytvořena jedna nebo více skupin studentů, stává se úloha pro kterou byla šablona definována otevřena. Z tohoto místa lze také vytvářet skupiny studentů pro jednotlivé šablony vypracování. Odkaz „Archiv“ zobrazí seznam úloh, jejichž řešení byla archivována, a po výběru úlohy také jednotlivá archivovaná řešení.

Všem uživatelům řídící aplikace přibyl v submenu Úlohy jeden odkaz pojmenovaný „školní úlohy“ (obrázek 5.1.1.b).



Obrázek 5.1.1.b – rozšíření menu (student)

Jestliže byl uživatel přiřazen do skupiny studentů, která má vypracovat některou šablonu vypracování, zobrazí se studentům ze skupiny po kliknutí na odkaz školní úlohy seznam částí vypracování, které by měli studenti vyřešit (případně také seznam již vyřešených a odevzdaných částí vypracování). Z tohoto místa si mohou studenti také zobrazit zadání parametrizace, která jim byla vybrána a také požadavky na přílohy k řešení, které by měli přiložit při odevzdávání části vypracování. Dále si pak mohou studenti zobrazit odevzdaná řešení a případně hodnocení pedagogem.

Odkaz aktivní úlohy vede na stránku zobrazující seznam rezervovaných úloh, které jsou právě aktivní a lze je vypracovat. Stránka aktivních úloh byla rozšířena tak, aby kontrolovala, jestli je zobrazovaná aktivní úloha také zadaná učitelem k vypracování. V případě, že má student rezervovanou úlohu, která je aktivní a která mu byla zadána k vypracování, systém studentovi zobrazí na stránce aktivních úloh také upozornění s následujícími informacemi:

1. název úlohy a části vypracování
2. datum do kdy má být část vypracování nejpozději odevzdána
3. odkaz na předchozí řešení (pokud bylo odevzdáno)
4. zadání parametrizace
5. odkaz na testovací a odevzdávací formulář

5.1.2. Definování testů a proměnných s technologií AJAX

Definování testů a proměnných se provádí na jedné stránce. Učitel vidí seznam vytvořených testů a také proměnných a může provést 6 operací – vytvořit, upravit nebo smazat test či proměnnou. Kdybych zpracovával požadavky učitele klasickou cestou, každý požadavek na vytvoření, úpravu či smazání by byl odeslán PHP stránce, která by operaci provedla, znova by načetla seznam všech testů a proměnných z databáze a výsledek odeslala učiteli.

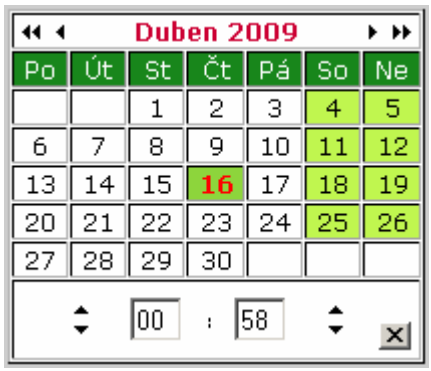
Z důvodů optimalizace jsem se rozhodl použít technologii AJAX [7]. Při vytváření, editaci či mazání testu nebo proměnné je prostřednictvím AJAXu zavolán obslužný PHP skript (ajax.php) na

serveru řídící aplikace, který požadovanou akci provede (volá metody příslušných managerů) a vrátí odpověď ve formátu XML. JavaScriptový kód, který je již součástí zobrazené stránky pro definování testů a proměnných, zpracuje odpověď na straně klienta a případně informuje učitele nebo změní obsah zobrazených informací na stránce. Tímto zpracováním požadavků na správu testů a proměnných nemusím refreshovat stránku při každém požadavku a tím také načítat seznam testů a proměnných z databáze.

Tuto technologii jsem využil na stránce pro správu testů a proměnných právě proto, že zde dochází také k zobrazování seznamů již vytvořených testů a proměnných, které by byly pokaždé načítány z databáze.

5.1.3. Komponenta datetimepicker

Pro zvýšení uživatelského komfortu při zadávání datumu a času začátku a konce části vypracování jsem využil externí komponenty datetimepicker [8]. Tato komponenta umožňuje pomocí graficky znázorněného kalendáře (obrázek 5.1.3) vybrat datum a čas. Zdrojový kód této komponenty se nachází v souboru datetimepicker_css.js.



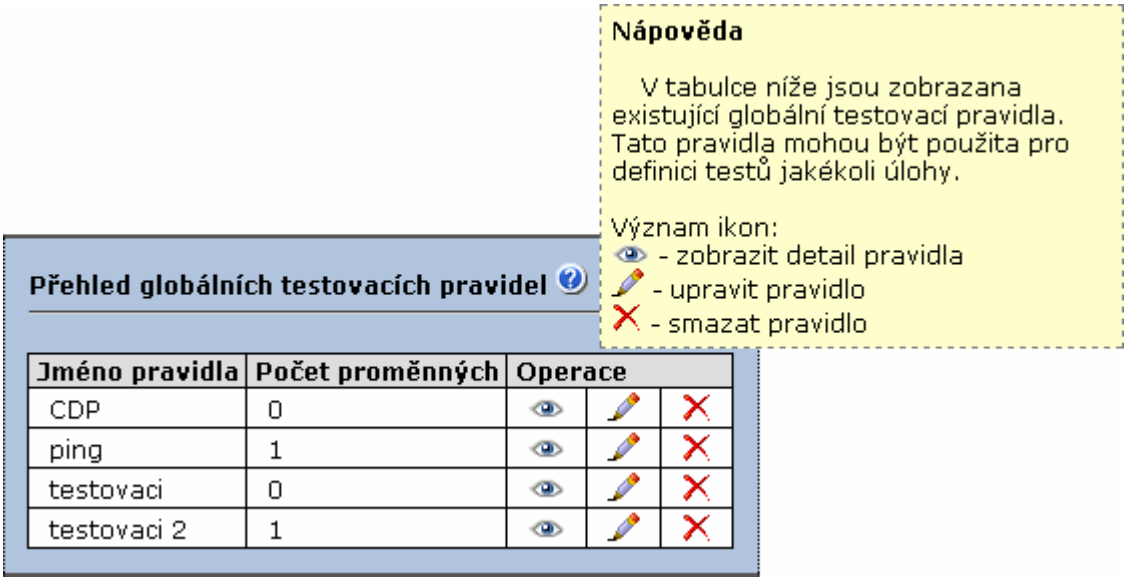
Obrázek 5.1.3 – Grafická podoba komponenty datetimepicker

5.1.4. Nápověda a upozornění uživatele

Pro zpříjemnění a usnadnění práce se systémem automatizace hodnocení konfigurací se na většině stránek uživatelského rozhraní nacházejí ikony umožňující uživateli zobrazit si nápovědu či upozornění související se zobrazenou stránkou. Uživatel se může setkat s těmito ikonami:

1) - Nápověda – zobrazí nápovědu
2) - Informace – zobrazí podrobnější informace
3) - Upozornění – zobrazí upozornění
4) - Chyba – zobrazí chybové hlášení
5) ☑ - Ok – slouží k vyjádření stavu „v pořádku“

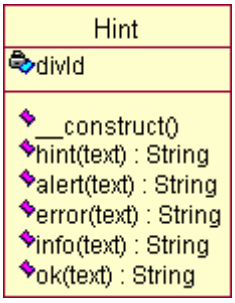
Pokud uživatel najede kurzorem myši nad ikonu, zobrazí se mu v okně prohlížeče v závislosti na typu ikony text nápovědy nebo text upozornění (informace, upozornění, chyba, ok). Příklad se zobrazenou nápovědou ukazuje obrázek 5.1.4.a.



Obrázek 5.1.4.a – Zobrazení nápovědy

Obsah nápovědy nebo upozornění se zobrazí ve žlutém poli. Toto textové pole může uživatel libovolně posouvat v rámci zobrazené stránky uvnitř prohlížeče.

Pro vkládání nápovědy na jednotlivé stránky slouží objekt Hint (obrázek 5.1.4.b). Volání vybrané metody zajistí vygenerování XHTML kódu potřebného pro vytvoření ikony a odkazu na zobrazení žlutého pole se zadaným textem (předaného parametrem metodě). Posouvání žlutého pole realizováno pomocí javascriptu (zdrojový kód v souboru hint.js) a grafické vyjádření zobrazeného pole je definováno pomocí kaskádových stylů (nacházející se v souboru testing.css).



Obrázek 5.1.4.b – třída popisující objekt Hint

6. Nasazení systému

Virtuální laboratoř disponuje několika virtuálními lokalitami sloužících pro vývojové a testovací účely. Virtuální lokalita představuje samostatně pracující lokalitu s plnohodnotným softwarovým vybavením, avšak fyzické síťové prvky jsou nahrazeny virtuálními. Pro testovací účely existuje několik virtuálních lokalit, které jsou navzájem propojeny jako produkční lokality distribuované virtuální laboratoře. Vytvořený systém automatizace hodnocení konfigurací byl nasazen a integrován do virtuální lokality Oslo. Webové rozhraní řídící aplikace této lokality lze nalézt na URL adrese: http://oslo.dvirtlab.net (přístupné pouze ze školní sítě VŠB-TUO).

Systém byl otestován a shledán funkčním. Systém umožňuje uživatelům provádět všechny funkce specifikované v kapitole 2. – Specifikace požadavků.

Rozmístění souborů se zdrojovými kódy a uživatelským rozhraním do souborového systému řídící aplikace je popsáno v příloze B – rozmístění zdrojových souborů.

Pro nasazení systému do lokality produkčního prostředí je potřeba udělat stejné kroky jako při nasazení systému do virtuální lokality, tedy:

a) databázi rozšířit o nové databázové tabulky a nastavit integritní omezení klíčů
b) do řídící aplikace dohrát nové zdrojové soubory
c) aktualizovat existující soubory řídící aplikace, které byly upraveny

Po domluvě s vedoucím práce bylo dohodnuto pilotní nasazení systému automatizace hodnocení konfigurací do produkčního prostředí a jeho využití při výuce předmětu Počítačové sítě v zimním semestru 09/10 na Vysoké škole báňské – Technické univerzitě Ostrava.

Závěr

Podle zadání diplomové práce se mi podařilo vytvořit softwarové dílo, které bylo specifikováno, navrženo a implementováno pomocí metod softwarového inženýrství. Vytvořený systém byl integrován do řídící aplikace Virtuální laboratoře a byl otestován.

Systém umožňuje pedagogům zadávat úlohy a semestrální projekty skupinám studentů, zadání úloh diferenciovat pro každou skupinu, vytvářet testy, pomocí kterých systém ověří správnost nakonfigurování topologie úlohy, a dále umožňuje hodnotit a archivovat odevzdaná řešení.

Používání systému pomůže vyučujícímu ušetřit čas, který by strávil při ručním opravování a vyhodnocování odevzdaných úloh a semestrálních projektů.

Řídící logika systému je implementována prostřednictvím objektového programování. V případě přeprogramování řídící aplikace za použití architektury MVC stačí k vytvořenému systému přidat komponentu controller. Komponenta model a view jsou již vytvořeny.

Systém automatizace hodnocení konfigurací bude nasazen do produkčního prostředí Distribuované virtuální laboratoře a bude prakticky využit při výuce předmětu Počítačových sítí v zimním semestru 2009/2010 na Vysoké škole báňské – Technické univerzitě Ostrava.

Literatura a informační zdroje:

[1] P. Němec: *Virtuální síťová laboratoř*. Diplomová práce, Ostrava, VŠB-TU FEI, 2005.

[2] J. Vavříček: *Rozvoj řídícího software virtuální laboratoře počítačových sítí*. Diplomová práce, Ostrava , VŠB-TU FEI, 2007.

[3] *POSIX Extended Regular Expressions* [online]. Dokument dostupný na URL: http://en.wikipedia.org/wiki/Regular_expression (květen 2009).

[4] J. ŠARMANOVÁ: *Teorie zpracování dat*. Ostrava, VŠB - Technická univerzita Ostrava, 1997. ISBN 80-7078-491-1.

[5] L. Ullman [přeložil Bogdan Kiszka]: *PHP a MySQL – Názorný průvodce tvorbou dynamických stránek*. Brno, Computer Press, 2004. ISBN 80-251-0063-4.

[6] *PHP* [online]. Dostupný dostupný na URL: http://cz2.php.net/manual/cs/index.php (květen 2009).

[7] R. Asleson, T. Schutta [přeložil Jakub Zemánek]: *AJAX – Vytváříme vysoce interaktivní webové aplikace*. Brno, Computer Press, 2006. ISBN 80-251-1285-3.

[8] *Javascript Date Time Picker 2.0* [online]. Dokument dostupný na URL: http://www.rainforestnet.com/datetimepicker.htm (květen 2009).

Příloha A: Datový slovník a integritní omezení

Datový slovník

Význam použitých symbolů a zkratek:
P – primární
C – cizí
A – ano
N – ne
ID – identifikátor

tr_archive_result				
sloupec	typ	klíč	nulový	popisek
id	int(11)	P	N	ID archivovaného řešení
idArchiveTask	int(11)	C	N	ID archivované úlohy
nameIration	varchar(45)		N	jméno části řešení
students	text		N	seznam studentů
done_on	datetime		N	datum a čas odevzdání
result_text	text		N	odevzdané řešení, výsledky testů a hodnocení

tr_archive_task				
sloupec	typ	klíč	nulový	popisek
id	int(11)	P	N	ID archivované úlohy
nameTask	text		N	jméno archivované úlohy
archived_on	datetime		N	datum a čas archivace

tr_chosen_test				
sloupec	typ	klíč	nulový	popisek
idIteration	int(11)	P, C	N	ID části vypracování
idTest	int(11)	P, C	N	ID testu

tr_enrolled_student				
sloupec	typ	klíč	nulový	popisek
idInstnace	int(11)	P, C	N	ID skupiny studentů
idUser	varchar(20)	P, C	N	ID uživatele

tr_filled_user_variable				
sloupec	typ	klíč	nulový	popisek

idInstance	int(11)	P, C	N	ID skupiny studentů
idVariale	int(11)	P, C	N	ID proměnné
value	text		N	hodnota vyplněné uživatelské proměnné

tr_filled_variable				
sloupec	typ	klíč	nulový	popisek
idParameterization	int(11)	P, C	N	ID parametrizace
idVariable	int(11)	P, C	N	ID proměnné
value	text		N	hodnota parametrizační proměnné

tr_instance				
sloupec	typ	klíč	nulový	popisek
id	int(11)	P	N	ID skupiny studentů
idTemplate	int(11)	C	N	ID šablony vypracování
idParameterization	int(11)	C	N	ID parametrizace

tr_instance_form				
sloupec	typ	klíč	nulový	popisek
idInstance	int(11)	P, C	N	ID skupiny studentů
given_by	varchar(20)	C	N	ID uživatele
given_on	datetime		N	datum vyplnění proměnných uložených do příštího vypracování

tr_iteration				
sloupec	typ	klíč	nulový	popisek
id	int(11)	P	N	ID části vypracování
idTemplate	int(11)	C	N	ID šablony vypracování
name	varchar(45)		N	jméno části vypracování
description	text		A	požadavky na přílohy k řešení
froma	datetime		N	datum a čas začátku části vypracování
toa	datetime		N	datum a čas konce části vypracování

tr_lock_testing				
sloupec	typ	klíč	nulový	popisek
idInstance	int(11)	P, C	N	ID skupiny studentů
endTime	datetime		N	datum a čas skončení testování

tr_parameterization				
sloupec	typ	klíč	nulový	popisek
id	int(11)	P	N	ID parametrizace
name	varchar(45)		N	jméno parametrizace

description	text		A	popisek parametrizace
idTask	int(11)	C	N	ID úlohy

tr_result				
sloupec	typ	klíč	nulový	popisek
idInstance	int(11)	P, C	N	ID skupiny studentů
idIteration	int(11)	P, C	N	ID části vypracování
done_on	datetime		N	datum a čas odevzdání řešení
result_student	text		N	odevzdané řešení s výsledky testování
result_teacher	text		N	podrobná zpráva testování
given_by	varchar(20)	C	N	ID uživatele, který řešení odevzdal
evalPoints	int(11)		A	bodové hodnocení řešení
evalText	text		A	textové hodnocení řešení

tr_script_variale				
sloupec	typ	klíč	nulový	popisek
code	varchar(22)	P	N	jméno (kód) proměnné ve skriptu
idTesting_rule	int(11)	P, C	N	ID testovacího pravidla
name	varchar(45)		N	uživatelsky příjemný název proměnné

tr_template				
sloupec	typ	klíč	nulový	popisek
id	int(11)	P	N	ID šablony vypracování
idTask	int(11)	C	N	ID úlohy
name	varchar(45)		N	jméno šablony vypracování

tr_test				
sloupec	typ	klíč	nulový	popisek
id	int(11)	P	N	ID testu
idTesting_rule	int(11)	C	N	ID testovacího pravidla
idTask	int(11)	C	N	ID úlohy
for_device	varchar(180)		N	jméno zařízení
points	int(11)		N	počet bodů za splnění testu
mandatory	tinyint(1)		N	povinnost pravidla

tr_testing_rule				
sloupec	typ	klíč	nulový	popisek
id	int(11)	P	N	ID testovacího pravidla
name	varchar(45)		N	jméno testovacího pravidla
script	text		N	skript pravidla
regular	text		N	vyhodnocovací regulární řetězec

marker	text		A	ukončovací značka (regulární výraz)
timeMarker	int		A	časová ukončovací značka
idTask	int(11)	C	N	ID úlohy

tr_used_variable				
sloupec	typ	klíč	nulový	popisek
idTest	int(11)	P, C	N	ID testu
code	varchar(45)	P, C	N	jméno (kód) proměnné ve skriptu
idVariable	int(11)	P, C	N	ID proměnné

tr_variabe				
sloupec	typ	klíč	nulový	popisek
id	int(11)	P	N	ID proměnné
idTask	int(11)	C	N	ID úlohy
name	varchar(45)		N	jméno proměnné
type	tinyint(1)		N	typ proměnné
description	text		A	popisek proměnné

Integritní omezení klíčů:

Vypsáno z databáze pomocí nástroje phpMyAdmin.

Omezení pro tabulku tr_archive_result:

```
ALTER TABLE `tr_archive_result`
  ADD CONSTRAINT `tr_archive_result_ibfk_1` FOREIGN KEY (`id`) REFERENCES
`tr_archive_task` (`id`) ON DELETE CASCADE;
```

Omezení pro tabulku tr_chosen_test:

```
ALTER TABLE `tr_chosen_test`
  ADD CONSTRAINT `tr_chosen_test_ibfk_2` FOREIGN KEY (`idTest`) REFERENCES
`tr_test` (`id`),
  ADD CONSTRAINT `tr_chosen_test_ibfk_1` FOREIGN KEY (`idIteration`)
REFERENCES `tr_iteration` (`id`) ON DELETE CASCADE;
```

Omezení pro tabulku tr_enrolled_student:

```
ALTER TABLE `tr_enrolled_student`
  ADD CONSTRAINT `tr_enrolled_student_ibfk_2` FOREIGN KEY (`idUser`)
REFERENCES `users` (`id`) ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `tr_enrolled_student_ibfk_1` FOREIGN KEY (`idInstance`)
REFERENCES `tr_instance` (`id`) ON DELETE CASCADE;
```

Omezení pro tabulku tr_filled_user_variable:

```
ALTER TABLE `tr_filled_user_variable`
  ADD CONSTRAINT `tr_filled_user_variable_ibfk_1` FOREIGN KEY
(`idInstance`) REFERENCES `tr_instance_form` (`idInstance`) ON DELETE
CASCADE,
```

```
  ADD CONSTRAINT `tr_filled_user_variable_ibfk_2` FOREIGN KEY
(`idVariable`) REFERENCES `tr_variable` (`id`) ON DELETE CASCADE;
```

Omezení pro tabulku tr_filled_variable:

```
ALTER TABLE `tr_filled_variable`
  ADD CONSTRAINT `tr_filled_variable_ibfk_2` FOREIGN KEY (`idVariable`)
REFERENCES `tr_variable` (`id`) ON DELETE CASCADE,
  ADD CONSTRAINT `tr_filled_variable_ibfk_1` FOREIGN KEY
(`idParameterization`) REFERENCES `tr_parameterization` (`id`) ON DELETE
CASCADE;
```

Omezení pro tabulku tr_instance:

```
ALTER TABLE `tr_instance`
  ADD CONSTRAINT `tr_instance_ibfk_2` FOREIGN KEY (`idParameterization`)
REFERENCES `tr_parameterization` (`id`) ON DELETE SET NULL,
  ADD CONSTRAINT `tr_instance_ibfk_1` FOREIGN KEY (`idTemplate`) REFERENCES
`tr_template` (`id`) ON DELETE CASCADE;
```

Omezení pro tabulku tr_instance_form:

```
ALTER TABLE `tr_instance_form`
  ADD CONSTRAINT `tr_instance_form_ibfk_2` FOREIGN KEY (`given_by`)
REFERENCES `users` (`id`) ON DELETE CASCADE,
  ADD CONSTRAINT `tr_instance_form_ibfk_1` FOREIGN KEY (`idInstance`)
REFERENCES `tr_instance` (`id`) ON DELETE CASCADE;
```

Omezení pro tabulku tr_iteration:

```
ALTER TABLE `tr_iteration`
  ADD CONSTRAINT `tr_iteration_ibfk_1` FOREIGN KEY (`idTemplate`)
REFERENCES `tr_template` (`id`) ON DELETE CASCADE;
```

Omezení pro tabulku tr_lock_testing:

```
ALTER TABLE `tr_lock_testing`
  ADD CONSTRAINT `tr_lock_testing_ibfk_1` FOREIGN KEY (`idInstance`)
REFERENCES `tr_instance` (`id`) ON DELETE CASCADE;
```

Omezení pro tabulku tr_parameterization:

```
ALTER TABLE `tr_parameterization`
  ADD CONSTRAINT `tr_parameterization_ibfk_1` FOREIGN KEY (`idTask`)
REFERENCES `tasks` (`id`) ON DELETE CASCADE ON UPDATE CASCADE;
```

Omezení pro tabulku tr_result:

```
ALTER TABLE `tr_result`
  ADD CONSTRAINT `tr_result_ibfk_7` FOREIGN KEY (`idInstance`) REFERENCES
`tr_instance` (`id`) ON DELETE CASCADE,
  ADD CONSTRAINT `tr_result_ibfk_8` FOREIGN KEY (`idIteration`) REFERENCES
`tr_iteration` (`id`),
  ADD CONSTRAINT `tr_result_ibfk_9` FOREIGN KEY (`given_by`) REFERENCES
`users` (`id`);
```

Omezení pro tabulku tr_script_variable:

```
ALTER TABLE `tr_script_variable`
```

```
  ADD CONSTRAINT `tr_script_variable_ibfk_1` FOREIGN KEY (`idTesting_rule`)
REFERENCES `tr_testing_rule` (`id`) ON DELETE CASCADE;
```

Omezení pro tabulku tr_template:

```
ALTER TABLE `tr_template`
  ADD CONSTRAINT `tr_template_ibfk_1` FOREIGN KEY (`idTask`) REFERENCES
`tasks` (`id`);
```

Omezení pro tabulku tr_test:

```
ALTER TABLE `tr_test`
  ADD CONSTRAINT `tr_test_ibfk_3` FOREIGN KEY (`idTesting_rule`) REFERENCES
`tr_testing_rule` (`id`) ON DELETE CASCADE,
  ADD CONSTRAINT `tr_test_ibfk_2` FOREIGN KEY (`idTask`) REFERENCES `tasks`
(`id`) ON DELETE CASCADE ON UPDATE CASCADE;
```

Omezení pro tabulku tr_used_variable:

```
ALTER TABLE `tr_used_variable`
  ADD CONSTRAINT `tr_used_variable_ibfk_8` FOREIGN KEY (`idVariable`)
REFERENCES `tr_variable` (`id`),
  ADD CONSTRAINT `tr_used_variable_ibfk_6` FOREIGN KEY (`idTest`)
REFERENCES `tr_test` (`id`) ON DELETE CASCADE,
  ADD CONSTRAINT `tr_used_variable_ibfk_7` FOREIGN KEY (`code`) REFERENCES
`tr_script_variable` (`code`);
```

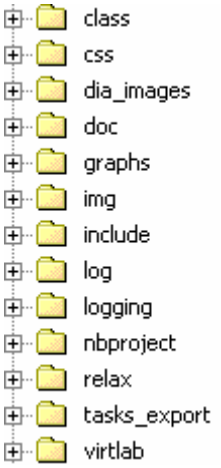
Omezení pro tabulku tr_variable:

```
ALTER TABLE `tr_variable`
  ADD CONSTRAINT `tr_variable_ibfk_1` FOREIGN KEY (`idTask`) REFERENCES
`tasks` (`id`) ON DELETE CASCADE ON UPDATE CASCADE;
```

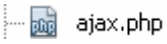
Příloha B – rozmístění zdrojových souborů

Soubory přidané do souborového systému řídící aplikace:

Adresářová struktura kořenové složky řídící aplikace:

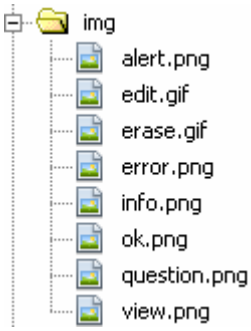


Soubory přidané do kořenové složky:

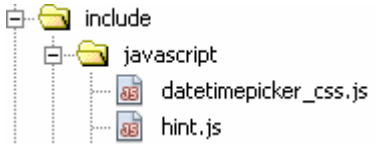


Soubory (kaskádové styly) přidané do složky css:

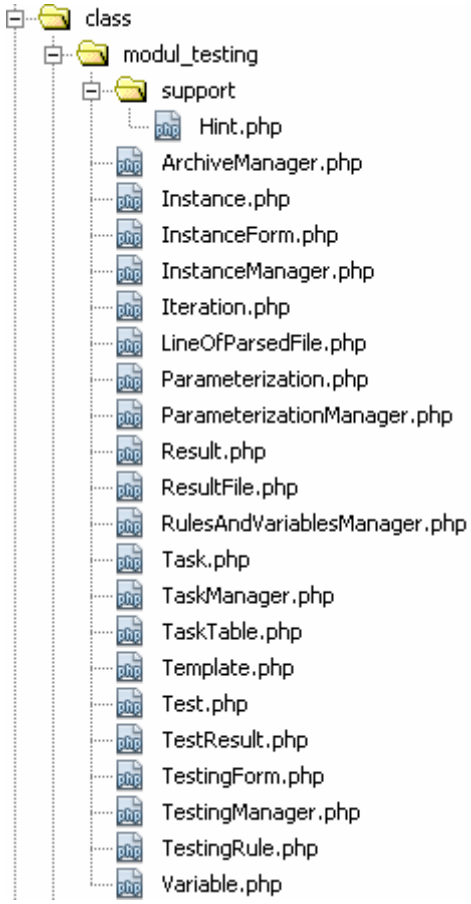
Soubory (obrázky) přidané do složky img:



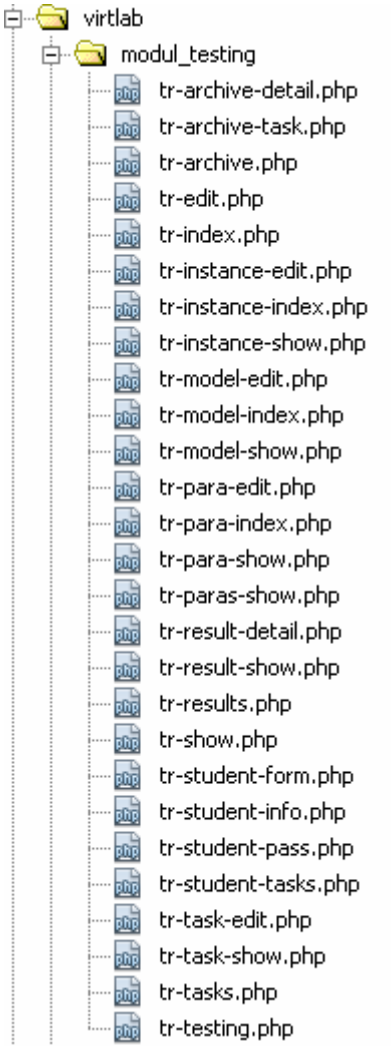
Soubory (kódy javascriptu) přidané do složky include/javascript:



Soubory (třídy) přidané do složky class:



Soubory (uživatelské rozhraní) přidané do složky virtlab:



Existující soubory řídící aplikace, které byly editovány:
/settings.php
/index.php
/class/virtlabWeb.php
/class/virtlabLanguage.php
/virtlab/menu.php
/virtlab/reser-active.php
/virtlab/reser-new.php

Příloha C

Obsah přiloženého CD

1. Text diplomové práce ve formátu PDF, soubor:
   Zdeněk Filipec - Automatizace hodnocení konfigurací - Diplomová práce.pdf
2. Text diplomové práce ve formátu DOC, soubor:
   Zdeněk Filipec - Automatizace hodnocení konfigurací - Diplomová práce.pdf
3. Zdrojové kódy vytvořeného systému, složka: source_files
4. Abstrakt a klíčová slova diplomové práce v češtině a angličtině, soubory:
   abstrakt a klíčová slova.txt
   abstract & keywords.txt