VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SYSTÉM PRO PREZENTACI A SHROMAŽĎOVÁNÍ DAT

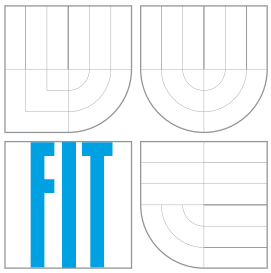
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAN SKÁCEL

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SYSTÉM PRO PREZENTACI A SHROMAŽĎOVÁNÍ DAT

SYSTEM FOR THE DATA PRESENTATION AND DATA RECEIVING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAN SKÁCEL

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MARTIN MILIČKA

BRNO 2013

Abstrakt

Cílem této bakalářské práce je navrhnout a implementovat nástroj umožňující sběr dat od více agentů a reprezentaci těchto dat přihlášeným uživatelům. Model zaznamenávaných dat je utvářen návrhářem pomocí speciálního nástroje integrovaného do vyvinuté aplikace. Backend aplikace je implementován v jazyce Python 2.7 za použití architektonického vzoru Model-view-controller. Použitým systémem řízení báze dat je MySQL. Pro definici jednotlivých pohledů v rámci aplikace je použit aplikačně specifický skriptovací programovací jazyk.

Abstract

The aim of this bachelor's thesis is to design and implement a tool that allows receiving data from multiple agents and representation of those data to logged in users. Model of recorded data is created by designer using special tool integrated into developed aplication. The application backend is implemented in Python language using Model-view-controller design pattern. Used database management system is MySQL. For definition of views within the application is used application specific scripting programming language.

Klíčová slova

IS, Informační systém, shromažďování dat, django, web, internet, služba

Keywords

IS, Information system, receiving of data, django, web, internet, service

Citace

Jan Skácel: Systém pro prezentaci a shromažďování dat, bakalářská práce, Brno, FIT VUT v Brně, 2013

Systém pro prezentaci a shromažďování dat

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Martina Miličky

.....
Jan Skácel
14. května 2013

Poděkování

Děkuji mé rodině za materiální i psychickou podporu. Dále děkuji vedoucímu mé práce za trpělivost a předané zkušenosti.

Obsah

Kapitola 1

Úvod

Před nástupem počítačů dominoval jako médium pro přenos a uchování dat papír. S překotným rozvojem informačních technologií a s ním spojeným rozvojem počítačových sítí se začaly rozvíjet elektronické informační systémy pro správu dat v řadě podniků a institucí. Tyto informační systémy mají oproti těm papírovým mnoho výhod. Jednou z nich je i možnost získávat data z různých vzdálených zdrojů automaticky, bez přispění člověka.

Cílem této práce je vyvinout webovou službu, skrze kterou uživatel může navrhnout webovou aplikaci v prostředí, které má WYSIWYG (what you see is what you get) podobu, ale dává uživateli i možnost vkládat konstrukce skriptovacího jazyka popisující aplikační logiku. Důraz je kladen na možnost vkládat data do systému automaticky ze vzdáleného zdroje, kterým může být například nějaké monitorované zařízení.

Mnoho elektronických informačních systémů má podobu webové aplikace. Pro vývoj webových aplikací vzniklo v minulosti již mnoho specializovaných jazyků i webových rámců (anglicky framework), které měly implementaci těchto aplikací zjednodušit či urychlit. Speciální kategorií webových rámců jsou takové, které nevyžadují znalost žádného programovacího jazyka a umožňují tvorbu webových aplikací stylem WYSIWYG. Existuje řada platforem pro WYSIWYG webové rámce fungujících na bázi SaaS (software as a service). Často umožňují nabídnout uživateli návrhové prostředí společně s platformou pro běh navržené aplikace. Typicky těmto službám schází možnost definovat jakoukoli aplikační logiku či perzistentně ukládat data.

Výsledná služba by měla kombinovat výhody rychlého vývoje ve WYSIWYG grafickém nástroji se sílou snadno rozšiřitelného textového programovacího jazyka. Komunikace od vzdáleného zdroje dat by mělo být možné uskutečnit pomocí protokolu XML-RPC nebo jednoduššího protokolu předání požadavku přes parametry v URL.

Kapitola 2

Použité technologie

V této kapitole je popis hlavních technologií a aplikací použitých pro potřeby této práce.

2.1 Python

Python je skriptovací objektově orientovaný programovací jazyk navržený v roce 1991 [11] Guido van Rossuem. Je vyvíjen jako open source projekt a pro většinu platforem je k dispozici zdarma. Python je dynamický více-paradigmatický jazyk.

Výkon interpretu CPython, který patří k nejpoužívanějším, je vysoký, neboť všechny kritické části jsou psány v jazyce C. Při výběru jazyka pro tento projekt bylo rozhodováno mezi vícero kandidáty. Jmenovitě pak PHP, Ruby a Python. Python byl zvolen, protože ze jmenovaných jazyků nejlépe odpovídá potřebám tvorby tak dynamických aplikací, jako je právě ta vyvinutá v rámci této práce. PHP například neumožňuje přímý přístup k aktuální tabulce symbolů. Jedním z mých cílů bylo postavit škálovatelnou aplikaci, která by mohla být potencionálně distribuována na více počítačích. Na to PHP není přizpůsobeno právě např. nemožností jednoduše ukládat session proměnné v relační databázi. Pro jazyk Ruby nejsou uzpůsobené ostatní navazující cloudové služby.

Ve vyvinuté aplikaci je použita verze jazyka 2.7 .

2.2 Webový framework Django

Django je open source webový aplikační rámec implementovaný v Pythonu, který se volně drží návrhového vzoru Model-view-controller [3] (česky Model-pohled-řadič). Je vydáván pod BSD licencí. Rámec je pojmenován po kytaristovi Django Reinhardtovi.

Návrhový vzor MVC je v rámci Django podpořen existencí bázové třídy Model, která je implicitně mapována na tabulku relační databáze [12]. Řadič v podání rámce Django je relace mezi množinou možných požadavků a funkcemi, které implementují pohledy. Implementace této relace je seznam dvojic regulárních výrazů a řetězců obsahujících názvy obslužných funkcí. Funkce implementující pohledy pak jako první parametr obdrží rozsáhlý objekt třídy request, který umožňuje přístup k proměnným webového serveru, cookies atp. Dalšími parametry jsou řetězce odpovídají pojmenovaným skupinám v regulárním výrazu, který byl použit pro mapovaní požadavku na daný pohled. Názvy skupin musí korespondovat s názvy parametrů funkce.

Django se řídí principem DRY („Don't repeat yourself“ tj. česky „Neopakuj se“)[12], který je viditelný například při definici nové třídy odvozené od bázové třídy Model. Poté,

co je taková třída definována, jsou příslušné tabulky v relační databázi utvořeny samy po zavolání příslušného příkazu. Programátor tedy nemusí definovat stejnou entitní množinu dvakrát. Objektově relační mapování je v Djangu automatizováno.

Webová aplikace postavená na rámci Django je implicitně složena z několika modulů, které jsou v kontextu terminologie Djanga nazývány aplikace. Tyto aplikace mají mezi sebou definováno rozhraní a lze je používat i napříč projekty. Tak je v rámci Django podpořena znovupoužitelnost kódu.

2.3 HTTP

HTTP (Hypertext Transfer Protocol) je síťový protokol sloužící k výměně hypertextových dokumentů ve formátu HTML. Používá obvykle port 80, verze 1.1 protokolu je definována v RFC 2616[2]. Jako protokol transportní vrstvy se často používá TCP[5].

V dnešní době je HTTP často používán i pro přenos jiných typů dokumentů a dat. Dokáže například přenést jakýkoliv soubor, který má definovaný MIME (podobně jako email). Dále je často používán společně s formátem XML pro tzv. webové služby (spouštění vzdálených aplikací) a zpřístupňuje i další protokoly, jako je např. FTP nebo SMTP. HTTP používá tzv. jednotný lokátor prostředků (URL, Uniform Resource Locator), který specifikuje jednoznačné umístění nějakého zdroje v Internetu[1].

Specifikace protokolu HTTP neumožňuje šifrování ani zabezpečení integrity dat. Pro zabezpečení HTTP se často používá TLS spojení nad TCP. Tato kombinace protokolů je označována jako HTTPS[6].

HTTP pracuje tak, že uživatel pošle serveru dotaz v textové formě, obsahujícího označení požadovaného dokumentu, informace o schopnostech prohlížeče apod. Server poté odpoví pomocí několika řádků textu popisujících výsledek dotazu (zda se dokument podařilo najít, jakého typu dokument je atd.), za kterými následují data samotného požadovaného dokumentu[2].

2.4 XML-RPC

XML-RPC pracuje na bázi zasílání HTTP dotazů na server, který tento protokol implementuje. Klient v takovém případě chce typicky zavolat jednu zvolenou metodu vzdáleného systému a obdržet její návratovou hodnotu.

Ve srovnání s protokoly třídy REST, kde jsou reprezentace zdrojů, tj. dokumentů pouze přenášeny, XML-RPC je navržen k volání metod.

2.5 Google app engine

Google app engine je platforma pro běh aplikací implementovaných v jazycích Python, Java nebo Go. Aplikace je uzavřena ve svém virtuálním prostředí, které umožňuje jen malé změny. Není například možná změna v systému souborů za běhu aplikace. Google app engine má v sobě zabudováno několik nerelačních databázových systémů, které ovšem tato práce nijak nevyužívá [8].

Google app engine dynamicky spouští a deaktivuje takový počet instancí interpretu aplikace, kolik jich je v dané chvíli potřeba. Umožňuje aplikacím snadno škálovat potřebný výpočetní výkon [8].

2.6 Google cloud storage

Google cloud storage je systém pro uložení větších objemů dat (například souborů). Data jsou organizovány po tzv. objektech, což jsou v terminologii Google cloud storage data spojená s metadaty, kterým je přiřazeno unikátní jméno. Objekty lze třídit do tzv. buckets. Ty mají unikátní jméno v rámci celého Google cloud storage [15].

2.7 Google cloud SQL

Je to databázové prostředí pro provoz MySQL nebo PostgreSQL serverů. Pro potřeby mé práce využívám instance MySQL systému verze 5 [4]. Administrátor má možnost editovat počet, výkon a paměťové prostředky virtualizovaných instancí systémů řízení bází dat. Dále je možné databáze zálohovat či importovat skrze webové rozhraní.

Kapitola 3

Analýza a návrh aplikace

3.1 Požadavky na aplikaci

Výsledná aplikace by měla mít následující kompetence:

- Výsledná aplikace by měla být schopna definovat model dat, se kterým pracuje
- Aplikace má mít možnost svá data číst a modifikovat
- Aplikace může spravovat přístup k těmto datům
  - Podle třídy uživatele
  - Podle jiných správcem systému definovaných pravidel
- Aplikace může data uživateli prezentovat takovou formou, kterou určí správce systému. Těmito formami mohou být například:
  - Grafy
  - Tabulky
  - Texty
- Systém může přijímat data od uživatel i od neživých agentů.

3.2 Shromažďování dat

Systém vytvořený uživatelem má schopnost přijímat data v HTTP komunikaci, které se mapují na proměnné přístupné při interpretaci pohledu. Jde zejména o data odeslaná formulářem metodou GET a POST. Metodu PUT nezpracovávají, ačkoli jedním z navržených rozšíření je implementace protokolu třídy REST, který tuto metodu využívá. Data odeslaná metodou POST mají to omezení, že formulář z něhož byla data odeslána musí být vygenerován v rámci interpretace některého z pohledů aplikace. Děje se tak v rámci ochrany aplikace před CSRF (cross site request forgery) útoky.

Speciální kategorií jsou pak data odeslaná protokolem XML-RPC. Pohled, který implementuje tento protokol se nachází na adrese URL_SYSTÉMU/xmlrpc. Jméno metody pak odpovídá jménu pohledu, který je následně interpretován za účelem vytvoření odpovědi. Parametry jsou v interpretu namapovány na proměnné, jejichž jméno dostaneme tak, že za

prefix „post_xmlrpc_“ vložíme celé číslo v desítkové soustavě odpovídající pořadí v jakém je tento parametr uveden ve volání metody.

Ve chvíli, kdy systém nabyl data, může je použít pouze pro vytvoření odpovědi. V takovém případě jsou ovšem ihned ztraceny. Pokud je systém chce uložit pro aktuální potřeby agenta, který je odeslal, může použít session proměnné. Ty lze vytvořit tak, že nově vzniklou proměnnou pojmenujeme s prefixem „session_“. Pro perzistentní uložení dat má každý systém k dispozici relační databázi.

3.3 Prezentace dat

Při přijetí dotazu zaslaného agentem je vytvoření odpovědi plně v kompetenci aplikační logiky navržené správcem systému. Pokud komunikace probíhá skrz webový prohlížeč, pak odpovědí může být libovolně formátovaný html dokument nebo xml dokument. Pokud se jedná o XML-RPC volání, návratovou hodnotou může být obsah libovolné proměnné. Aplikace podporuje zobrazování některých typů dat.

Existuje možnost podmínit přístup k některým datům příslušností uživatele k některé z předem definovaných tříd. Tento mechanismus nechrání ovšem přímo data, nýbrž mechanismus, kterým se k datům přistupuje. Jiný přístupový mechanismus může tyto data zobrazit. Občas můžeme stejný pohled zobrazit různým třídám uživatel jinak, podle jejich potřeb. Například v případě skladové evidence může účetní hledat jiná data, než skladník.

Shromážděná data lze prezentovat strukturovaným textem, tabulkami či grafy. Aplikace napojena na rozhraní Google charts API, skrze které lze snadno vytvářet grafy, které čerpají informace z dat, které jsou zaslány s odpovědí.

3.4 Agenti a kanály, kterými komunikují

Agentem rozumíme celek schopný samostatného jednání v závislosti na okolních podnětech. V kontextu této práce můžeme rozlišit dvě třídy agentů komunikujících se systémy běžícími v rámci vyvinuté aplikace:

- Živí agenti

  Těmi rozumíme uživatele, kteří komunikují skrz svůj počítač.

- Neživí agenti

  Jedná se o obecné programy, které dokáží odesílat data přes XML-RPC nebo HTTP, za předpokladu dodržení pravidel, které systém předepisuje.

Uživatelé mohou komunikovat se systémem obdobně jako mohou s většinou webových aplikací - pomocí svého prohlížeče a HTTP. Podobně jako je irelevantní operační systém a druh prohlížeče při dodržení standardního protokolu a popř. jazyka pro popis odpovědi (HTML, XML atd.), tak je irelevantní i jazyk v němž je vytvořen neživý agent.

3.5 Použité návrhové vzory

Návrhový vzor je obecným znovu použitelným řešením problému. Je to pomůcka, která má zabránit znovu-vyvíjení stejného řešení a zároveň má zrychlit porozumění zdrojového kódu jiným programátorem.

3.5.1 MVC

Návrhový vzor Model-view-controller je volně používán ve webovém rámci Django. Návrhový vzor rozděluje aplikaci do tří hlavních částí.

Model

Model je reprezentace informací s nimiž aplikace pracuje. V aplikací je to skupina tříd, které dědí od třídy Model.

View

Převádí data poskytovaná modelem do vhodné podoby, která je předána uživateli. V aplikaci je to funkce, která vkládá data vytvořená controllerem do šablony a vytváří odpověď na požadavek uživatele.

Controller

Reaguje na události a aktualizuje model. V aplikaci je to interpret jazyka navrženého v rámci této aplikace pro popis logiky uživatelem spravovaných systémů.

3.5.2 Lazy initialization

Dalším návrhovým vzorem, který je využit v aplikaci je Lazy initialization. Tento návrhový vzor odkládá vytvoření objektu či provedení určitého výpočtu do doby vzniku požadavku na jeho výstup. V aplikaci je tento návrhový vzor užit při dotazování databáze.

3.6 Datový model aplikace

Datový model aplikace lze rozdělit na dvě části. První je reprezentace dat užívaných pro provoz aplikace, správu uživatelem vytvořených systémů, pohledů definovaných správci systémů apod. Druhou částí jsou databáze (schémata) pro data specifická pro každý uživatelem vytvořený systém.

3.6.1 Model provozních dat aplikace

Provozní data aplikace jsou spravována v jediné databázi (viz obr. 3.1). Pro modelování databáze spravující provozní data byly identifikovány následující entity:

- Systém

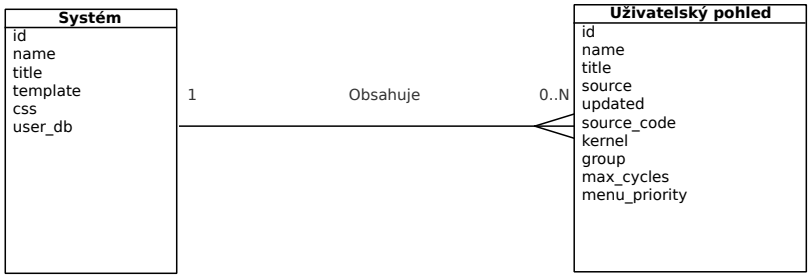
  Systém reprezentuje uživatelem vytvořený systém.

- Pohled (View)

  Pohled je pak reprezentace definice aplikační logiky sestavující odpověď na uživatelem zaslaný požadavek.

Ostatní entitní množiny (uživatelé, skupiny uživatelů apod.) zde není potřeba reprezentovat, neboť si je každý uživatelem vytvořený systém může spravovat sám. Portál s registrací a administrací pro správce systémů je také uživatelský systém, který má ovšem rozšířené kompetence. Je tak schopen modifikovat, utvářet i mazat jiné systémy. Tohoto

je docíleno speciálními jazykovými konstrukcemi v definici pohledů tohoto systému. Tyto konstrukce nelze používat v jiných systémech.



Obrázek 3.1: ERD databáze provozních dat. Tabulky přesně odpovídají entitním množinám. Cizí klíče nejsou uvedeny.

3.6.2 Automatické generování schématu ze zdrojového kódu

Django dokáže z tříd, které dědí od třídy `model` vytvořit jejich reprezentaci v relační databázi. Schéma databáze se vytváří a aktualizuje příkazem `python manage.py syncdb` spuštěným v kořenovém adresáři Django projektu. Zde je potřeba si povšimnout, která databáze je v souboru s nastavením uvedena jako 'default'.

3.6.3 Model dat uživatelských systémů

Schéma uživatelského systému (viz obr. 3.2) je v rukou správce každého systému. Při vytvoření nového systému je pro potřeby správy práv pro editaci pohledů vytvořeno několik tabulek, které nelze odstranit, ale lze je doplnit o další sloupce. Těmito tabulkami jsou:

- users

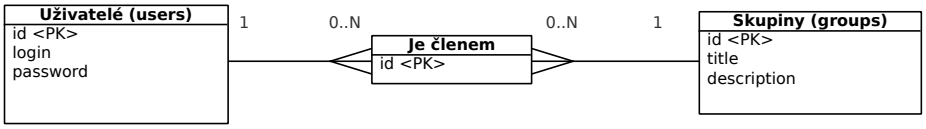
  Tabulka users obsahuje reprezentaci uživatel systému.

- groups

  Tabulka groups obsahuje skupiny uživatel.

- user_groups

  Entitní množina uživatel a skupin jsou ve vztahu N ku N, tedy každý uživatel může být ve více skupinách a jedna skupina může obsahovat více uživatel. Pro potřeby modelování tohoto vztahu je tedy vytvořena další tabulka user_groups.



Obrázek 3.2: Schéma databáze vytvořené při registraci nového systému. Cizí klíče nejsou uvedeny.

3.7 Uživatelské rozhraní

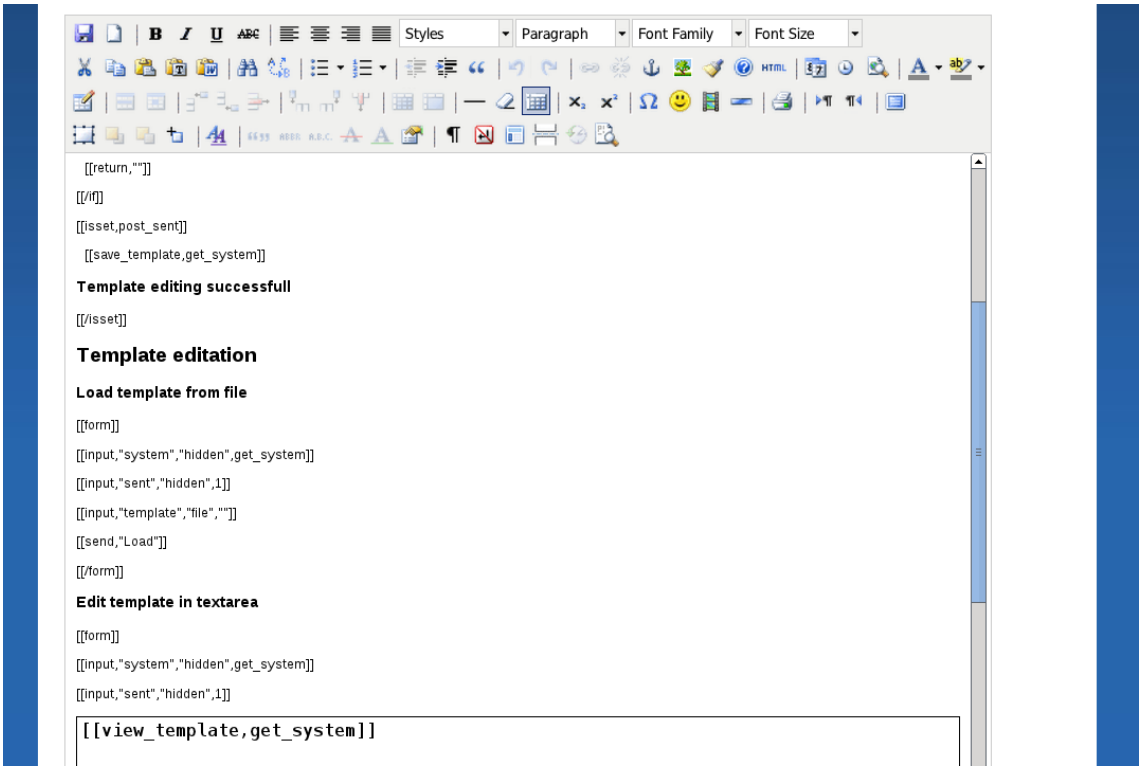
Uživatelské rozhraní aplikace je realizováno skrz webový portál, který je zároveň jedním ze systémů, které umožňuje spravovat. Tento portál umožňuje registraci nových uživatel, přihlášení, vytváření systémů, úpravu HTML šablony systémů a změnu hesla uživatele root u každého systému.

3.8 Jazyk pro popis uživatelských pohledů

Každý pohled (webová stránka) uživatelského systému má definovaný zdrojový kód v jazyce, který byl navržen pro tento účel. Uživatel může editovat pohledy svého systému skrze WYSIWYG rozhraním (viz obr. 3.3), ke kterému je použita knihovna TinyMCE. Kromě textu, různých prvků a jejich formátování lze ovšem vkládat i jazykové konstrukce, které mají speciální sémantiku. Lze tak například vypsat text `'Ahoj světe'` konstrukcí `[[print,'Ahoj světe']]`. Pokud následně tento příkaz označíte a nastavíte formátování na tučné či určíte tento příkaz jako nadpis, v editaci tento příkaz uvidíte skutečně tučně či větší než zbytek textu. Při interpretaci pohledu pak výstup tohoto příkazu bude obalen tagy b, h1 apod. (viz obr. 3.4).

3.8.1 Návrh jazyka

Kombinace WYSIWYG editoru HTML a textového programovacího jazyka představuje výzvu. Pro potřeby této práce jsem definoval požadavky, které by měl splňovat takový jazyk. Hlavním požadavkem je syntaktická ortogonalita s HTML a obecně s XML. Ta je dosažena tím, že pro ohraničení konstrukcí nejsou užívány tagy. Dále by měl jazyk usnadňovat definici částí informačních systémů, které se s různými obměnami často implementují (formuláře a jejich zpracování, generování grafů apod.). Důležitým požadavkem byla i podpora bezpečnosti, která je realizována například automatickým vkládáním CSRF tokenů do formulářů. Tato ochrana proti podvržení formulářů není pro uživatele na první pohled viditelná.



Obrázek 3.3: Editace pohledu odpovědného za editaci šablony v hlavním systému.

Obrázek 3.4: Interpretace pohledu odpovědného za editaci šablony v hlavním systému.

3.8.2 Gramatika jazyka pro popis uživatelských pohledů

Aplikace umožňuje vytváření aplikační logiky tak, aby uživatel mohl vytvořit libovolný informační systém. Do jazyka pro popis této logiky jsou zabudovány konstrukty pro popis přihlášení, odhlášení, správu přístupu, správu relační databáze, generování grafů, zpracování formulářů apod. Každá jazyková konstrukce začíná řetězcem „[[“ a končí „]]“. Mezi „[[“ a „]]“ jsou parametry oddělené čárkou. Prvním parametrem je jméno konstrukce, které určuje její účel.

Obecný příklad: `[[JMÉNO_KONSTRUKCE,PARAMETR_1,...,PARAMETR_N]]`. Některé konstrukce jsou párové. Obalují tak sekvenci jiných konstrukcí nebo textu, který je beze změny převeden na výstup. Blok konstrukce se ukončuje příkazem: `[[/JMÉNO_KONSTRUKCE]]`, což je podobné tomu jak fungují tagy v HTML či XML dokumentech.

Jazyk byl navrhován jako regulární. Chyby nesouhlasících si počátků a konců párových konstrukcí jsou při překladu do mezikódu odhalovány při sémantické analýze. Chyby parametrů jsou odhalovány až za běhu kvůli možnému dynamickému sémantice některých konstrukcí. Řetězce „[[“ a „]]“ používané pro ohraničení konstrukcí lze změnit v nastavení aplikace (tedy v souboru settings.py).

3.8.3 Sémantika jazyka

Parametry konstrukcí jsou (až na několik zdokumentovaných výjimek) výrazy jazyka Python. Ty jsou přímo interpretovány po nahrazení proměnných konstantními literály. Interpretace se provádí v režimu znemožňující volání jiných než předdefinovaných funkcí (`len()`,`int()` apod.). Je tak znemožněno možnému bezpečnostnímu problému známému jako Code injection.

Každá konstrukce odpovídá jedné třídě ve zdrojovém kódu. Tato třída dědí od bázové třídy jazykových konstrukcí. Třídním atributem se udává jméno konstrukce. Speciální metodou se pak definuje, zda je daná instance konstrukce párová. Dalšími metodami jsou `open()` a `close()`, které jsou volány při otevření konstrukce a při jejím uzavření.

Pro ukázku je zde uvedeno několik příkladů. V prvním příkladu je demonstrováno nastavování hodnot proměnných. První příkaz nastaví proměnnou `hello` na hodnotu `'Hello world!'`. Druhý příkaz přiřadí do proměnné `foo` hodnotu celočíselného literálu 12. Proměnná `foo` je sečtena s číselným literálem 13.0 a přiřazena do proměnné `bar`. Proměnná `bar` je nyní typu číslo s plovoucí desetinnou čárkou.

```
[[set,hello,'Hello world!']]
[[set,foo,12]]
[[set,bar,foo+13.0]]
```

Následující příklad je ukázkou definice jednoduchého rozhraní. V tomto případě se jedná nejspíše o registrační formulář. První a poslední příkaz tvoří párovou konstrukci `form`. Tato konstrukce je mimo jiné odpovědná za ochranu formulářů proti CSRF. Do výstupu tedy vkládá neviditelný prvek tzv. CSRF token. Další nepárové konstrukce uvnitř konstrukce `form` definují jednotlivé prvky rozhraní. Vidíme zde konstrukce třídy `text` a `input`. Obě tyto konstrukce na výstupu tvoří html tag input. Konstrukce `text` vytváří tag input s atributem type nastaveným na hodnotu text. Atribut name tohoto tagu je nastaven na stejnou hodnotu, jaká je uvedena v prvním parametru. Konstrukce `send` vytvoří tlačítko pro odeslání formuláře. Má nepovinný parametr, který určuje popisek tohoto tlačítka. Pokud tento popisek není definovat je použit text „submit“.

```
[[form]]
[[text,'username']]
[[input,'mypassword1','password','']]
[[input,'mypassword2','password','']]
[[send]]
[[/form]]
```

Poslední uvedený příklad demonstruje čtení dat z databáze a jejich prezentaci uživateli. Je zde dobré připomenout, že editace zdrojových kódů probíhá částečně ve WYSIWYG režimu a tagy, uvedené v příkladu nejsou v implicitním zobrazení vidět. Uživatel si ovšem může zobrazit kód v textovém režimu a mnoho změn je tak možné dělat pohodlněji. V příkladu níže si můžeme povšimnout dvou použitých konstrukcí. První z nich je `for` a druhá je `print`. Konstrukce `for` zajišťuje cyklické provádění příkazů do splnění ukončující podmínky. Nicméně každá iterace tohoto cyklu se děje pro prvek entitní množiny jejíž jméno je udáno prvním parametrem. Dalšími nepovinnými parametry jsou podmínky, které musí každý vybraný prvek splňovat. V příkladu níže se jedná o to, aby atribut `name` obsahoval text „Bender“. Konstrukce `print` zajišťuje vložení hodnoty výrazu dané prvním parametrem do výstupu. Konstrukce `for` inicializuje a nastavuje proměnné obsahující hodnoty jednotlivých atributů entity, pro kterou se daná iterace vykonává. Jméno každé této proměnné utvořeno konkatenací jména entitní množiny, znaku podtržítka a jména atributu.

```
[[for,'People',name contain 'Bender']]
<div class='FuturamaCast'>
[[print,'Id: '+People_id]]<br/>
[[print,'Name: 'People_name]]<br/>
[[print,'Adress: 'People_address]]<br/>
</div>
[[/for]]
```

3.9 Bezpečnost

Při návrhu aplikace byla vynaložena velká snaha, aby sebou aplikace nenesla větší než nezbytné množství zranitelností. Vzhledem k tomu, že aplikační logika jednotlivých uživatelských systémů je v rukou jejich správců, není pro to tedy moc prostoru. Jako hrozby, které je ze strany aplikace možné eliminovat byly identifikovány CSRF, SQL injection a code injection zranitelnosti.

3.9.1 CSRF zranitelnosti

CSRF je zkratka z Cross site request forgery. Zranitelnost spočívá ve vytvoření podvrženého formuláře, kterého útočník přiměje legitimního uživatele odeslat. V minulosti bylo navrženo a implementováno několik obranných opatření. Mezi těmi, které byly uvažovány jako vhodní kandidáti pro aplikaci vyvinutou v rámci této práce byly:

- Vyžadování tajného, unikátního tokenu (řetězce náhodných znaků) ve všech odeslaných formulářích. Takto je docíleno toho, že útočník není schopen tento token vložit do svých podvržených formulářů[9].
- Zkontrolovat, že hlavička každého požadavku obsahuje X-Requested-With nebo kontrola HTTP Referer, a nebo HTTP Origin[10]. Tato obranná strategie byly prokázaná jako nespolehlivá[14][13].

Ve světle nabytých znalostí bylo provedeno rozhodnutí chránit aplikaci před výše uvedeným typem útoků přes unikátní token (skrytý prvek s nahodilým obsahem) v každém z odeslaných formulářů. Webový rámec Django pro toto již má implementovanou podporu. Ta spočívá hlavně v kryptograficky kvalitním generátoru pseudonáhodných čísel. Je totiž očividně důležité, aby další generovaný token nešel odhadnout útočníkem (mohl by jím vytvořit podržený formulář, který by byl opatřen platným tokenem).

Prvek s tokenem je vložen vždy když se interpretuje konstrukce `[[form]]`. V ideálním případě si uživatel nemusí být ani vědom, že systém touto ochranou disponuje.

3.9.2 SQL injection

SQL injection je forma útoku, kdy vložením speciálně vytvořeného vstupu útočník přistoupí k datům, ke kterým nemá mít přístup nebo provede neoprávněnou operaci na datové vrstvě aplikace. SQL injection je speciálním případem code injection útoku. Zranitelný je takový software z jehož vstupů jsou nesprávně filtrovány únikové znaky pro řetězcové literály zabudované v SQL.

V aplikaci vyvinuté v rámci této práce je obrana proti této třídě útoků implementována tak, že každý dotaz na databázi je ošetřen funkcí, která upravuje literály řetězců tak, že do nich na vhodná místa vkládá únikové sekvence znaků. Tato funkce je distribuována v balíku implementujícím rozhraní z databází. Existují další řetězce jako například názvy tabulek, které nelze touto funkcí ošetřovat. Tyto vstupy jsou validovány pomocí regulárních výrazů nepovolujících žádné speciální znaky.

3.9.3 Code injection

Speciálním případem tohoto typu útoku je SQL injection. Jedná se o vložení vstupu, který je na straně serveru interpretován jako kód v místě, kde to není očekáváno. Výrazy v zabudovaném programovacím jazyce jsou interpretovány přímo Pythonem, proto je tato zranitelnost rizikem, proti kterému je potřeba se nějak jistit.

Obranou implementovanou u aplikace vyvinuté v rámci této práce je znemožnění volání jakékoliv metody objektu přes tečkovou notaci, kontrola únikových sekvencí u řetězcových literálů a změna tabulky symbolů v místě interpretace tak, aby byly přístupné jen základní vestavěné funkce.

3.10 Adresářová struktura

Vzhledem k tomu, že aplikace byla vyvinuta nad webovým rámcem Django, adresářová struktura projektu tak byla poměrně jasně daná. V adresáři s aplikací najdeme 2 soubory a pět podadresářů. V relativních cestách k souborům budeme v této kapitole tento adresář považovat jako výchozí (`./`)

- ./manage.py

  Soubor `./manage.py` je vygenerovaný Djangem při vytvoření projektu. Ovládá se skrz něj testovací server a provádí se aktualizace a instalace schématu relačního modelu do databáze.

- ./app.yaml

  V souboru `./app.yaml` se nachází nastavení WSGI (Web Server Gateway Interface) rozhraní používané na produkčních Google app engine serverech.

- Adresář s nastavením (./ilumexis)

  Adresář stejného jména jako je jméno aplikace (v případě instance běžící na herobe.com jde o jméno „ilumexis“) obsahuje konfigurační soubory aplikace. Najdeme v něm soubory `./ilumexis/settings.py` s nastavením a `./ilumexis/urls.py` mapující url na funkce přiřazující požadavkům odpovědi.

- ./boto

  Adresář `./boto` obsahuje modul externí knihovny boto, který není na Google apps engine serverech přítomen, a kterého je využíváno pro spojení s datovým skladem pro ukládání dynamických souborů (vkládaných a měněných uživateli systémů).

- ./templates

  Adresář `./templates` obsahuje Django šablony. Tyto šablony obalují šablony vkládané správci při registraci nových systému.

- ./dcp

  Adresář `./dcp` obsahuje autorem této práce implementovaný kód. Jméno adresáře je zkratkou z „data collecting and presentation“. Tento adresář je Python modulem a zároveň Django aplikací. V Django frameworku se vzniklé programy nazývají projekty, které se sestávají z více aplikací, které v standardní terminologii odpovídají pojmu modul. Kód je v tomto adresáři rozdělen do šesti souborů.

  - __init__.py

    Tento soubor je prázdný. Jeho existence sděluje interpretu jazyka Python, že adresář obsahuje modul.

  - models.py

    Tento soubor obsahuje modely webového rámce Django. Z tříd modelů entit jsou extrahovány informace pro tvorbu schématu relační databáze. Tuto činnost provádí skript obsažený ve webovém rámci Django. Dále je zde třída `dbms` implementující rozhraní s relační databází MySQL verze 5. Úpravou metod této třídy lze docílit napojení celé aplikace na jiný databázový systém než MySQL.

– views.py

  V tomto souboru lze nalézt řadu funkcí implementujících jednotlivé pohledy podle mapování z `./ilumexis/urls.py`. Prvním parametrem těchto funkcí je objekt požadavku, dalšími mohou být řetězce extrahované z URL.

– ilumis_interpret.py

  Zde se nachází třída pro správu připojení k Google cloud storage a třída interpretu jazyka pro popis pohledů uživatelských systémů. Z hlediska využití dynamických vlastností jazyka Python je zajímavá funkce `scan_for_constructs()`. Tato funkce má totiž za úkol najít všechny třídy, které dědí od třídy `lang_construct`. Jsou to třídy jazykových konstrukcí vestavěného programovacího jazyka pro popis pohledů uživatelských systémů. Asociativní pole těchto tříd a jmen konstrukcí použitých jako klíče je poté použito k inicializaci třídního atributu třídy `lang_interpret`. Tím se docílí toho, že jazykové konstrukce jsou vyhledávány jen jednou a to při načítání modulu do paměti. Díky tomu odpadá nutnost vedení záznamů o každé existující jazykové konstrukci.

– constructs_base.py

  Nachází se zde bázová třída jazykových konstrukcí. Zajišťuje, že každá třída, která od této třídy dědí má minimální rozsah metod. Třída, která od ní dědí se automaticky stává nepárovou jazykovou konstrukcí s žádnou sémantikou. Toto lze měnit pomocí přetěžování metod či parametrů. Na příklad jméno konstrukce se určuje přetížením příslušného třídního atributu a párovost přetížením virtuální metody, jejích návratová hodnota je kýženým predikátem.

– constructs.py

  V tomto souboru se nachází implementace tříd jednotlivých jazykových konstrukcí. Celkově je uživatelům, potažmo správcům jednotlivých systémů přístupno 26 jazykových konstrukcí. Dále je implementována série konstrukcí, které jsou užívány pro správu provozní databáze, tedy vytváření nových systémů, načítání zdrojových kódu pro editaci pohledů atp.. Třetí a poslední kategorií konstruktů jsou takové, které jsou použitelné všude, ale jsou určeny pro zobrazení informací pro odstraňování chyb nebo hledání optimalizací systému. Manuál dokumentuje pouze první kategorii.

3.11 Tvorba odpovědi na požadavek uživatele

Po zaslání požadavku HTTP protokolem se nejprve zjistí, zda ip adresa zdroje není zakázaná (např. v důsledku minulých dos útoků apod.), poté je zjišťováno, zda požadavek sebou nenese informace, které vyžadují kontrolu CSRF tokenu. Pokud tyto kontroly proběhnou v pořádku, aktivuje se systém pro vyhledání příslušného Django pohledu. Ve chvíli, kdy je nalezen příslušný pohled, je zavolána funkce implementující tento pohled. Pokud je požadavek zaslán v rámci protokolu XML-RPC, deaktivuje se CSRF kontrola. Následně aplikace zjistí o který systém a uživatelský pohled se jedná, interpretuje ho, vloží výsledek0 do šablony systému a odešle kompletní odpověď zpět tazateli.

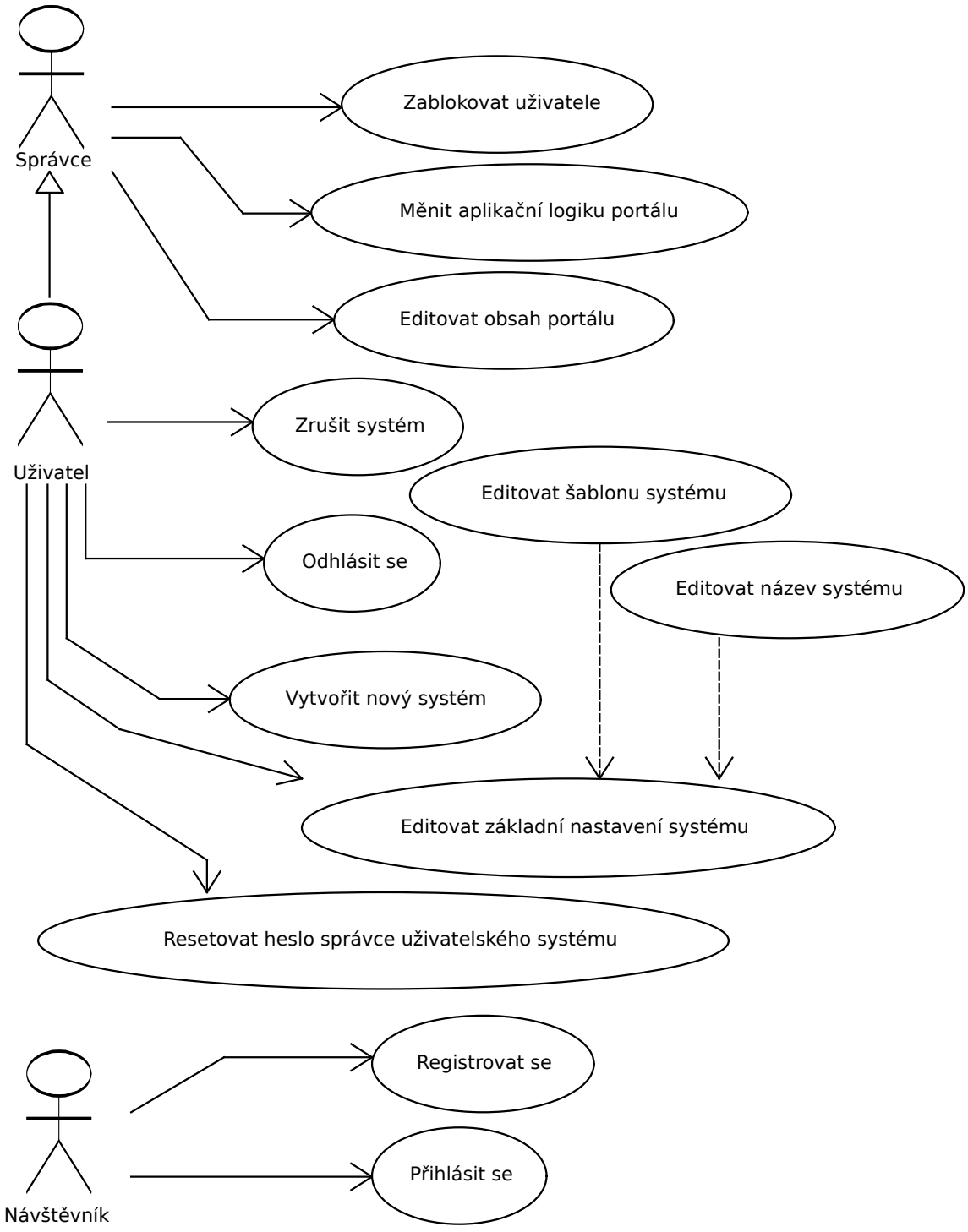
Za povšimnutí stojí to, že ochrana proti CSRF útokům je v režimu odpovědi na XML-RPC požadavek deaktivovaná. Jsou pro to dva hlavní důvody. U XML-RPC požadavku je předpoklad, že přiložená data nemusí odpovídat dříve vygenerovanému formuláři a druhým důvodem je to, že jediná data předávaná interpretu jsou nepojmenované parametry volané procedury (implementována jako pohled uživatelského systému).

3.12 Portál a uživatelské systémy

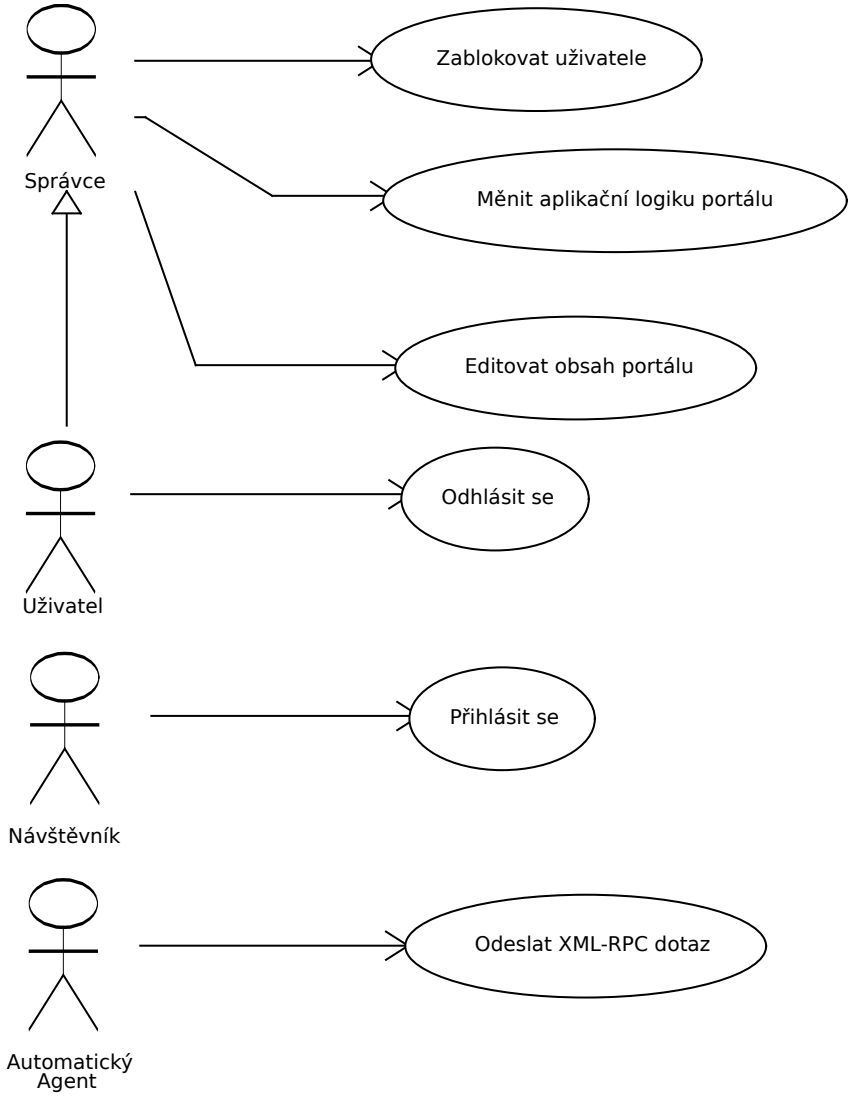
Celá aplikace běžící na serveru by se dala charakterizovat jako interpret jazyka, kterým jsou popsány jednotlivé uživatelské systémy. Řídící systém, ve kterém je přítomna registrace a administrace nových správců a jejich systémů je pro potřeby této práce nazýván jako portál nebo hlavní systém. Ostatní systémy, které běží na jiné subdoméně než www jsou v této práci nazývány uživatelské systémy. Hlavní systém není nijak odlišný od těch systémů, které mají ve správě jednotliví správci. Má k dispozici speciální jazykové konstrukce jimiž může například vytvářet nové systémy a dělat jiné činnosti, které uživatelské systémy dělat nemohou. Dále jsou jeho obsahem speciální pohledy, na které se odkazuje v každém uživatelském systému. Jsou to například pohledy pro přihlášení či odhlášení uživatel, editaci databáze či pohled pro editaci pohledů. Zde je zajímavostí, že poslední jmenovaný pohled může za jistých okolností editovat sám sebe.

Pro vizualizaci akcí, které mohou uživatelé konat v rámci portálu byl vytvořen diagram užití (viz obr. 3.5). U uživatelských systémů je obsah tohoto diagramu odvislý od aplikační logiky, kterou do nich vloží jejich správci. Minimální diagram užití pro uživatelské systémy lze také nalézt v této práci (viz obr. 3.5). Druhý diagram užití je podmnožinou prvního.

Toto řešení bylo zvoleno z několika důvodů. Prvním byla možnost otestovat si množství z nabízených schopností systému již při jeho vývoji. Aby různé aspekty hlavního systému správně fungovali, musí správně fungovat i skriptovací jazyk, na kterém systém běží. Druhým důvodem byla praktičnost takového řešení. Cílem celé práce bylo urychlit vývoj různých typů webových aplikací a díky tomuto přístupu bylo možné využít různých vlastností vyvinuté aplikace už k jejímu vlastnímu vývoji. Systém je tak možné při objevu chyby opravit velmi rychle. Stejně tak je možné rychle doplnit požadovanou funkcionalitu. Pokud se jedná o změnu funkcionality, která zasahuje samotný skriptovací jazyk, je potřeba nutné provést úpravu ve zdrojovém kódu aplikace.



Obrázek 3.5: Diagram užití hlavního systému



Obrázek 3.6: Minimální diagram užití uživatelského systému

Kapitola 4

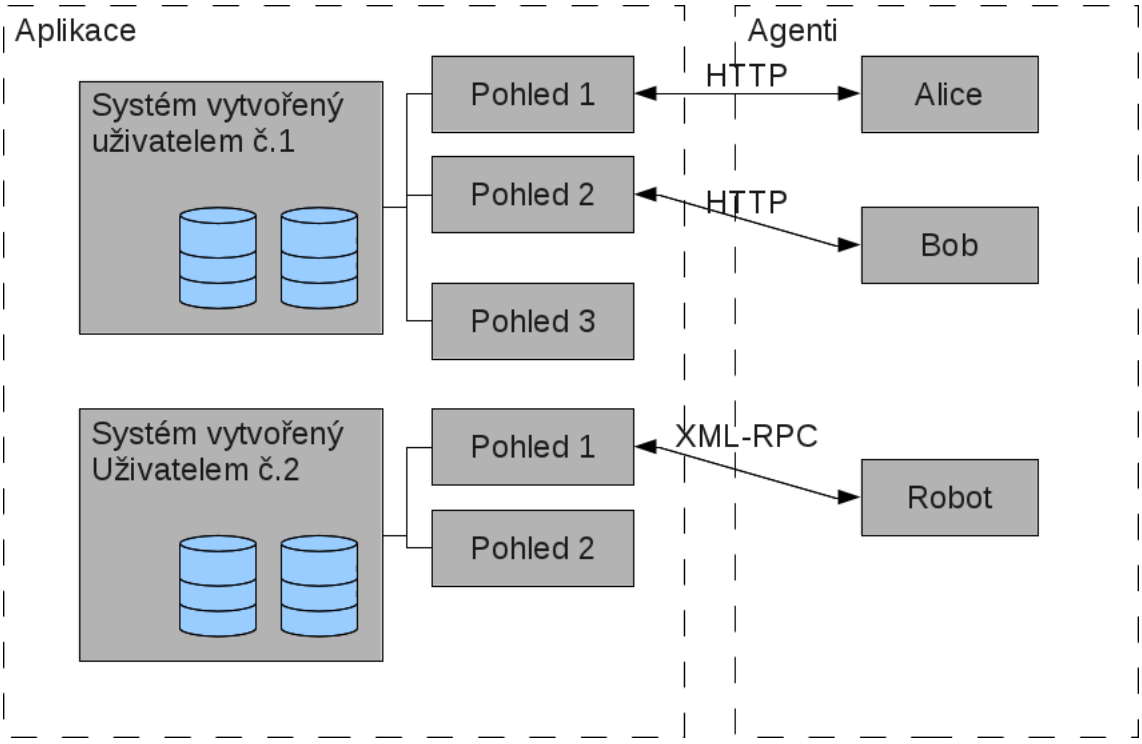
Implementace

4.1 Infrastruktura

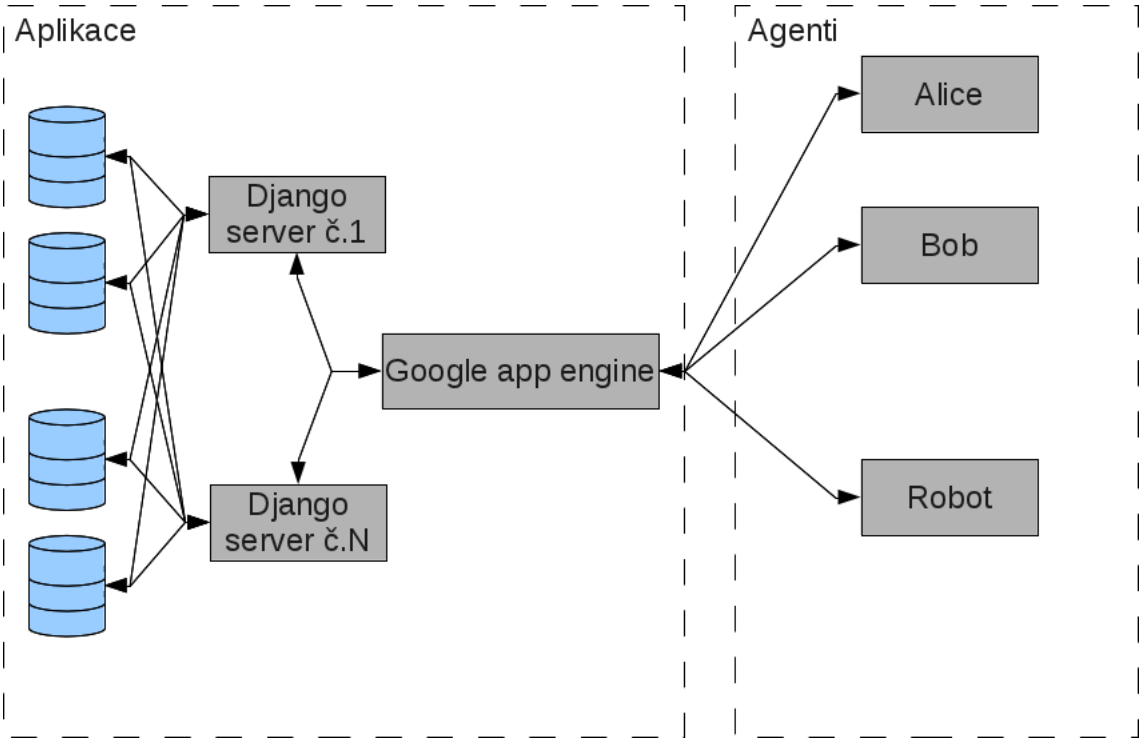
Pro implementaci byla vybrána skupina služeb od společnosti Google. Google app engine je systém, který dynamicky (podle stanoveného limitu v latenci mezi příchodem dotazů a odpovědí na ně) spouští další instance interpretu aplikace [8]. Tyto interprety běží každý na svém virtuálním stroji, jehož základní parametry lze nastavit (počet cyklů virtuálního procesoru za sekundu, velikost RAM). Počet těchto interpretů není omezen, ačkoli čas jejich běhu je účtován. Aby náročnost některých opakujících se výpočtů nebyla neúnosná, Google app engine umožňuje aplikaci využít rozsáhlý memcache server. Aplikace vyvinutá v rámci této práce této cache široce využívá zejména pro zvýšení výkonnosti uložiště souborů.

Další služba, kterou aplikace využívá je Google cloud SQL. Při návrhu byla uvažována řada databázových služeb společnosti Google. Mezi nimi bylo i několik nerelačních databází. Z důvodů snadné přenositelnosti kódu byla zvolena služba Google cloud SQL, která spočívá v pronájmu virtuálního serveru s běžící databází MySQL. Podobně jako u instancí v Google app engine i zde lze nastavovat výkon jednotlivých serverů. Google cloud SQL nabízí řadu nástrojů pro snadné zálohování nebo například pro vytvoření replik pro zvýšení celkového výkonu databázového systému [4]. Při spuštění aplikace v instanci GAE (Google app engine) dojde k inicializaci spojení na všechny dostupné databázové systémy [12]. Tyto spojení jsou udržovány perzistentně (viz obr. 4.2).

Poslední služba, kterou aplikace vyvinutá v rámci této práce využívá ke svému běhu, je Google cloud storage. Jedná se o virtualizované uložiště dat. Data lze strukturovat do dvou úrovní. V první úrovni jsou tzv. buckets. Lze si je představit jako adresáře bez možnosti obsahovat podadresáře. V nich se nachází entity obsahující data společně s jejich metadaty. Těmto entitám se v kontextu Google cloud storage říká objekty. Pojem objekt zde lze snadno zaměnit za pojem soubor. Díky nemožnosti data strukturovat do entit podobným adresářům bylo nutné přijít s řešením, jak tento zjevný nedostatek obejít. Existují dva principiální řešení. Jednodušším z nich je celou cestu zapsat do jména objektu. Problém nastává při přejmenování adresáře. V takovém případě je potřeba přejmenovat všechny koncové soubory. Druhým řešením je použít jako jméno objektu libovolný unikátní řetězec, pro který bude existovat záznam v jiné reprezentaci adresářové struktury. Tou je obvykle nějaká forma relační databáze. Pro druhé řešení existuje podpora ze strany webového rámce Django. Současná podoba aplikace používá první řešení z výše popsaných.



Obrázek 4.1: Schéma aplikace z pohledu uživatele.



Obrázek 4.2: Schéma aplikace z pohledu architektury.

4.2 Škálovatelnost

Škálovatelnost neboli rozšiřitelnost v softwarovém inženýrství je žádoucí vlastnost systému, sítě nebo procesu, která má schopnost pracovat s náhlými změnami potřeby obsluhy čili zvyšovat sledované parametry v případě, že nastane taková potřeba. Vzhledem k rozsahu vyvinutého systému je zde popsaná škálovatelnost jednotlivý částí systému. Při výběru infrastruktury hrála její škálovatelnost velkou roli. Proto bylo zvoleno plně virtualizované prostředí, kde lze snadno přidávat nové výpočetní prostředky a nastavovat jejich výkon.

Interprety aplikace jsou, jak již bylo zmíněno dříve, spouštěny dynamicky dle potřeby. Kdyby bylo možné vytvořit odpověď na dotaz bez součinnosti s ostatními částmi (memcache, databáze, storage) byla by škálovatelnost celé aplikace prakticky neomezená. Respektive by ji omezovali prostředky Google cloudu a samozřejmě rozpočtu organizace provozující tuto aplikaci. Každopádně i z provedených měření vyplývá, že tato vrstva není úzkým hrdlem aplikace. Tj. že Google app engine neomezuje rychlost běhu aplikace tak, jako jiné subsystémy.

Databáze jsou tradičně nejhůře škálovatelnou komponentou webových aplikací. Pomocí virtuálního prostředí Google cloud SQL lze docílit toho, že jednotlivým databázovým serverům nedojde paměť [4]. Pokud je některý z databázových systémů přetížen, lze přistoupit k vytvoření jeho repliky nebo replik. Tyto repliky díky předřazenému vyvažovači zátěže způsobí zvýšení rychlosti zpracování dotazů při nichž dochází ke čtení z databáze. Problém nastává ve chvíli, kdy databázový systém nestíhá zapisovat nebo aktualizovat data v paměti. Pro takovou situaci není vytvořen scénář pro reakci. Aplikace má oddělený databázový systém (nejen schemata) pro provozní data a uživatelské systémy. Toto je primárně bezpečnostní opatření, nicméně zvyšuje i škálovatelnost ve smyslu, že je možné pro každý systém vytvořit vlastní uzavřené databázové prostředí tvořené i více než jedním systémem (viz obr. 4.2). Ačkoli je zřejmé, že toto by bylo nákladné. Hlavní nedostatek škálovatelnosti těchto databázových systémů je, že jejich vytváření a nastavování není automatizované, a že řešení případného přetížení by muselo být vyřešeno ručně. To je velký rozdíl oproti Google app engine, který toto řeší automaticky [8].

4.3 Konvence dodržované při implementaci

Zdrojové kódy byly do značné míry vytvářeny dle konvence jazyka Python PEP 8. Odlišností je například nerozdělování privátních a veřejných metod prvním znakem jejich jména (u privátních se občas udává znak '_' na počátek jejich jména). Toto je z důvodů přehlednosti, protože během vývoje se často stávalo, že metoda, která byla považována za striktně interní bylo potřeba vyjímečně volat. Na konec byl učiněn závěr, že toto rozdělování nemá vliv na výslednou přehlednost kódu, a že je v tomto směru mnohem důležitější kvalitně dokumentovat činnost jednotlivých metod, aby se předešlo nepředpokládaným důsledkům volání.

Přístupové funkce nejsou ve většině případů implementovány. Výjimku tvoří přístupové funkce k výsledku interpretace pohledu. K některým atributům je přistupováno přímo z důvodů optimalizace kódu. Například přístupy k atributu obsahující slovník hodnot proměnných a jejich názvů (tabulka symbolů interpretu pohledů) je takovým atributem, jelikož se k němu přistupuje velice často a zprostředkování tohoto atributu přes rozhraní metody by znamenalo zbytečné zdržení.

4.4 Automaticky generovaná dokumentace

Všechny komentáře ve zdrojovém kódu jsou tvořeny tak, aby byly kompatibilní s aplikací Epydoc pro automatické generování dokumentace. Komentáře ve zdrojovém kódu jsou z důvodu lepší přenositelnosti a znovupoužitelnosti psány v anglickém jazyce. Takto zdokumentovány jsou téměř všechny funkce. U každé funkce, metody či procedury, která je komentována je popsána její činnost, parametry, typy parametrů a návratová hodnota (jeli přítomna). Popsány jsou dále třídy a atributy. Další odlišností je občasné nedodržování doporučení výskytu maximálně 79 znaků na řádku. Často to bylo způsobeno dlouhými názvy prvků použitých v některých výrazech. Jedná se o kompromis mezi přehledností kódu z hlediska pojmenovávání entit, odsazování a dodržováním standardů.

Speciální kategorií jsou metody tříd implementující funkcionalitu jednotlivých jazykových konstrukcí vestavěného programovacího jazyka. Komentována je jejich bázová třída, od které všechny výše jmenované třídy dědí. Metody potomků jsou až na výjimky tvořeny přetížením virtuálních metod bázové třídy. Jejich činnost se příliš nemění. Není tedy potřeba mít více než třicet stejných komentářů k výše popsaným metodám. Tyto metody jsou tedy cíleně neokomentovány. Jednotlivé jazykové konstrukce jsou popsány v manuálu k vestavěnému programovacímu jazyku (ten je přítomen na webu projektu).

4.5 Implementace automatických agentů

Automatickým agentem míníme obecný program, který komunikuje se systémem vytvořeným uživatelem. Vzhledem k tomu, že celá aplikace je psána v jazyce Python 2.7, byl tento jazyk zvolen pro demonstraci minimálního programu, odesílajícího data do vzdáleného systému:

```
import sys
import xmlrpclib
rpc_srv = xmlrpclib.ServerProxy('http://example.herobe.com/xmlrpc')
result = rpc_srv.insert_user('Bender','benderbendingrodrigez@futurama.com')

if result == '1':
  print('Success')
elif result == '2':
  print('Name too short')
else:
  print('Error')
```

Pro pochopení tohoto programu jsou klíčový třetí a čtvrtý řádek. Na třetím řádku je vytvořen objekt pro komunikaci se systémem běžícím na doméně `example.herobe.com`. Na čtvrtém řádku je zavolána metoda `insert_user`. Jméno metody je stejné jako jméno obslužného pohledu v cílovém systému. V tomto případě tedy platí, že pokud bychom s tímto obslužným pohledem chtěli komunikovat skrze HTTP, nalezli bychom jej na URL `http://example.herobe.com/insert_user`. Návratovou hodnotou je textová reprezentace toho, co vytvořil pohled `insert_user` jako parametr konstrukce `return`. V případě, že takovou konstrukci pohled neobsahuje je výsledná hodnota prázdná. V případě Pythonu se jedná o objekt None.

Odeslaná data jsou ve výše uvedeném příkladě dva řetězce a to konkrétně `Bender` a `benderbendingrodrigez@futurama.com`. Parametry jsou mapovány na proměnné dostupné pro interpret pohledu. Jejich jméno je `post_xmlrpc_N`, kde `N` je celé číslo, které představuje pořadí v jakém byly zadány při volání metody. První parametr má `N` rovno 0, další má `N` rovno 1 atd. Systém dokáže pracovat se třemi datovými typy a neomezeným počtem parametrů. Těmi třemi typy jsou celé číslo (integer), číslo s plovoucí desetinnou čárkou a řetězec.

Kapitola 5

Možná rozšíření

Možná rozšíření lze rozdělit do několika základních oblastí. Rozšíření vestavěného programovacího jazyka, vytvoření podpory dalších rozhraní pro komunikaci s agenty a definice logiky aplikace pomocí dalších nástrojů, než zmíněného programovacího jazyka.

5.1 Rozšíření vestavěného programovacího jazyka

Vestavěný programovací jazyk obsahuje v této chvíli přes 3 desítky konstrukcí. Neobsahuje možnosti abstrakce částí kódu (funkce, procedury, objekty atp.) pro opakované použití. Možnost definice takových abstrakcí či jen maker by mohlo být velkým přínosem pro uživatele.

Jazyk byl tvořen tak, aby bylo možné snadno a rychle přidat nové konstrukce, pokud po nich vznikne potřeba. V průběhu beta testování byla tvorba nových konstrukcí jedním z důležitých cílů i proto, že původní myšlenka byla taková, že systém by měl obsahovat velký počet konstrukcí implementující velice úzkou specifickou činnost. V tomto se tento jazyk do značné míry liší od jiných moderních programovacích jazyků, které se snaží docílit spíše menšího množství konstrukcí, jejichž sekvence se dají různě abstrahovat.

Jazyk v této chvíli nepodporuje ternární konstrukce (typu if-then-else). Konstrukce byly navrhovány jako unární nebo párové. Ačkoli toto nepředstavuje problém co do vyjadřovací síly jazyka, často by existence takových konstrukcí mohla zrychlit vývoj nebo zpřehlednit kód. Jedním z navržených řešení je vytvoření konstrukce vkládané dovnitř bloku obaleném párovou konstrukcí. Tato nová konstrukce by blok rozdělila na dva a v součinnosti s párovou by utvořila funkční celek. Pro příklad můžeme uvést párovou konstrukci if, která spustí interpretaci bloku za splnění podmínky předanou jako parametry. Novou konstrukcí, by mohla být konstrukce else. Iniciativa by ovšem musela být stále na konstrukci if, která by při nesplnění podmínky musela ověřit, zda blok uvnitř neobsahuje ještě konstrukci else. Tento příklad je ještě o to komplikovanější, že je potřeba ověřit, že případný else náleží právě do daného páru konstrukce if.

V současné podobě jazyka nejsou přítomny konstrukce pro efektivní manipulaci s řetězci. Dobrým rozšířením by byla možnost pracovat s regulárními výrazy. Jazyk Python má velmi kvalitní standardní modul `re`, pro který by jistě šlo vytvořit jazykové rozhraní. Problém, který nastal, když s tímto bylo experimentováno tkví v neodhadnutelné výpočetní náročnosti takového regulárního výrazu. Nevhodně napsaný regulární výraz by mohl být dobrým vektorem pro DOS útok. Nicméně přínos takového rozhraní je dostatečný, aby o něm bylo vážně uvažováno.

5.2 Podpora dalších rozhraní

V této chvíli lze se systémy utvořenými uživateli komunikovat skrze webové rozhraní, XML-RPC a předáváním parametrů v URL (metoda GET) či v hlavičce požadavku (metoda POST). Při návrhu aplikace byly zvažovány další formy komunikace. Konkrétně nějaká forma REST a SOAP API.

5.2.1 REST

REST je zkratkou z Representational State Transfer. Je to styl softwarové architektury pro distribuované systémy propojené počítačovou sítí jako je například internet. REST byl navržen jako styl rozhraní pro libovolné webové API.

Termín Representational State Transfer byl poprvé použit a definován Royem Fieldingem v jeho doktorské disertační práci v roce 2000. Fielding je mimo jiné jedním z autorů specifikace HTTP verze 1.0 a 1.1[7].

REST komunikuje s webovou aplikací pomocí metod požadavků GET, POST, PUT a DELETE. Jednou z podmínek, kterou si tento protokol klade je bezstavovost. Na dva po sobě jdoucí stejné dotazy musí tedy být vytvořeny stejné odpovědi. To je ve chvíli, kdy se odpovědi generují podle aplikační logiky, která je v rukou správce uživatelského systému velmi komplikované. Řešením by mohlo být vytvoření speciálního rozhraní, které by splňovalo všechny podmínky na něj kladené aby se mohlo označovat RESTful (v souladu s požadavky kladené na protokol navržený podle pravidel REST)[7].

5.2.2 SOAP

Termín SOAP byl původně zkratkou z Simple Object Access Protocol. Je to protokol pro výměnu strukturovaných informací webových služeb v počítačových sítích. Jednotlivé zprávy jsou XML (Extensible Markup Language) dokumenty[16]. Pro posílání a přijímaní SOAP zpráv se obvykle používá HTTP (Hypertext Transfer Protocol) nebo STMP (Simple Mail Transfer Protocol)[16].

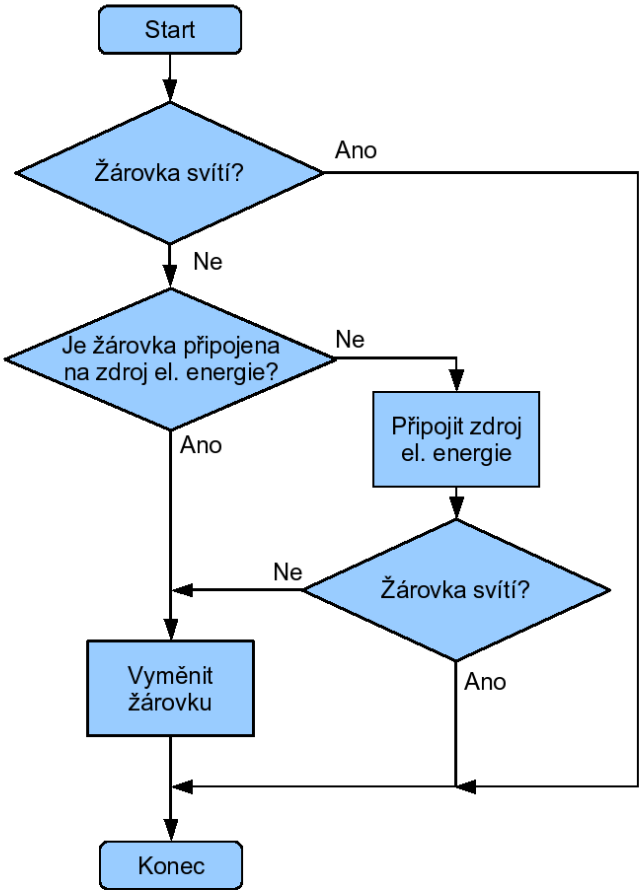
SOAP je považován za nástupce XML-RPC. XML-RPC rozhraní je již v rámci této práce implementováno. SOAP je tedy logickým rozšířením. SOAP přenáší celou řadu informací a plnohodnotná implementace by mohla být ve výsledku nepřehledná v jejím napojení na interpret pohledů.

5.3 Definice logiky systémů pomocí dalších nástrojů

Aplikační logika uživateli vytvořených systémů je definována ve zdrojovém kódu každého z pohledů těchto systémů. Je možné, že by bylo účelné vytvořit další možnost vykonávání logiky systému někde mimo pohledy. Způsob definice této logiky by se mohla výrazně lišit od té, jíž jsou definovány pohledy. Uvažovány byly Vývojové diagramy, Petriho sítě. Kromě těchto způsobů definice logiky systémů byla uvažována i alternativa, kdy by se využil jeden ze standardizovaných formátů pro možné importování těchto definic navržených v jiných, externích nástrojích. Jako takový formát by mohl sloužit například XMI (XML Metadata Interchange).

5.3.1 Vývojové diagramy

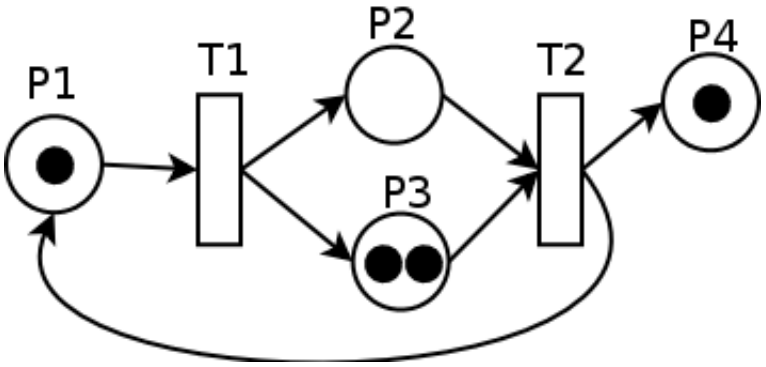
Vývojové diagramy jsou používány k návrhu a dokumentaci procesů nebo programů. Podobně jako u jiných typů diagramů, pomáhají některé děje vizualizovat a usnadnit jejich uživateli pochopit nějaký proces (viz obr. 5.1). Existuje mnoho různých druhů vývojových diagramů používající různé boxy, notace a konvence.



Obrázek 5.1: Ukázka vývojového digramu pro výměnu žárovky

5.3.2 Petriho sítě

Petriho sítě představují matematický jazyk pro popis paralelních procesů. Existuje velké množství různých druhů Petriho sítí. Jedná se o druh orientovaných bipartitních grafů, ve kterém kruhy značí místa a svislé obdélníky brány. Podle daného druhu Petriho sítě se řada věcí může lišit (jak jsou definovány podmínky průchodu místy, strategie vyhodnocení atp.) (viz obr. 5.2).



Obrázek 5.2: Jednoduchá Petriho síť. Autor: Máté Soós

Kapitola 6

Reálné použití a testování

6.1 Testování

6.1.1 Testování sémantiky jazykových konstrukcí

Pro účely testování funkčnosti všech zdokumentovaných jazykových konstrukcí byly vytvořeny speciální stránka, která cíleně obsahovala všechny konstrukce. Dohromady prováděla jednoduchý aritmetický výpočet, který následně vypsala. Operandy tvořili parametry získané metodou GET i POST. Operandy byly zapsány do relační databáze a zase z ní extrahovány. Kromě tohoto byla přímo testována i ochrana proti CSRF, pokusem o podvržení formuláře.

Výsledkem bylo odhalení několika chyb, kdy se konstrukce chovali v rozporu s návrhem a dokumentací. Takto odhalené chyby byly následně opraveny.

6.1.2 Testování rozhraní mezi systémy a automatickými agenty

Tyto testy měly za úkol verifikovat chování XML-RPC rozhraní v primitivních scénářích. Celkově se jedná o tři testy popsané níže. Pohled `test` zpracovává požadavky od agentů, provádějících testy. Kód pohledu je následující:

```
[[if,post_xmlrpc_0 == ,,test1‘‘]]
  [[return,post_xmlrpc_1]]
[[/if]]

[[if,post_xmlrpc_0 == ,,test2‘‘]]
  [[return,post_xmlrpc_1 + post_xmlrpc_2]]
[[/if]]
```

Následující testy lze nalézt na přiloženém CD, kde jsou přítomny v jediném skriptu.

Test odeslání dat agentem a přijetí jejich kopie

Kód agenta:

```
# TEST 1
import sys
import xmlrpclib
from time import gmtime, strftime

rpc_srv = xmlrpclib.ServerProxy(,,http://www.herobe.com/xmlrpc‘‘)
current_time = strftime(,,%a, %d %b %Y %H:%M:%S +0000‘‘, gmtime())
result = rpc_srv.test(,,test1‘‘,current_time)

if result == current_time:
  print(,,TEST 1: Request sent successfully‘‘)
else:
  print(,,TEST 1: Error‘‘)
```

Činnost tohoto kódu je taková, že odešle zformátovaný aktuální čas v podobě řetězce a očekává přijetí stejného řetězce zpět. V případě, že se vygenerovaný a přijatý řetězec shoduje, test neodhalil chybu.

Test odeslání jednoduchého zadání agentem a přijetí výsledku

Kód agenta:

```
# TEST 2
import sys
import xmlrpclib
from time import gmtime, strftime

rpc_srv = xmlrpclib.ServerProxy(,,http://www.herobe.com/xmlrpc‘‘)
a = 2
b = 3
result = rpc_srv.test(,,test2‘‘,a,b)

if result == str(a+b):
  print(,,TEST 2: Request sent successfully‘‘)
else:
  print(,,TEST 2: Error‘‘)
```

Činnost tohoto kódu je taková, že odešle hodnoty proměnných `a` a `b` a přijme hodnotu, která by měla být jejich součtem. V případě, že se součet vypočtený na straně agenta shoduje s přijatou hodnotou, test neodhalil chybu.

Test reakce systému na chybný název funkce (pohledu)

Kód agenta:

```
# TEST 3
import sys
import xmlrpclib
from time import gmtime, strftime

rpc_srv = xmlrpclib.ServerProxy(,,http://www.herobe.com/xmlrpc‘‘)
try:
result = rpc_srv.nonexistent()
print(,,TEST3: Error‘‘)
except xmlrpclib.Fault:
print(,,TEST3: Request sent successfully‘‘)
```

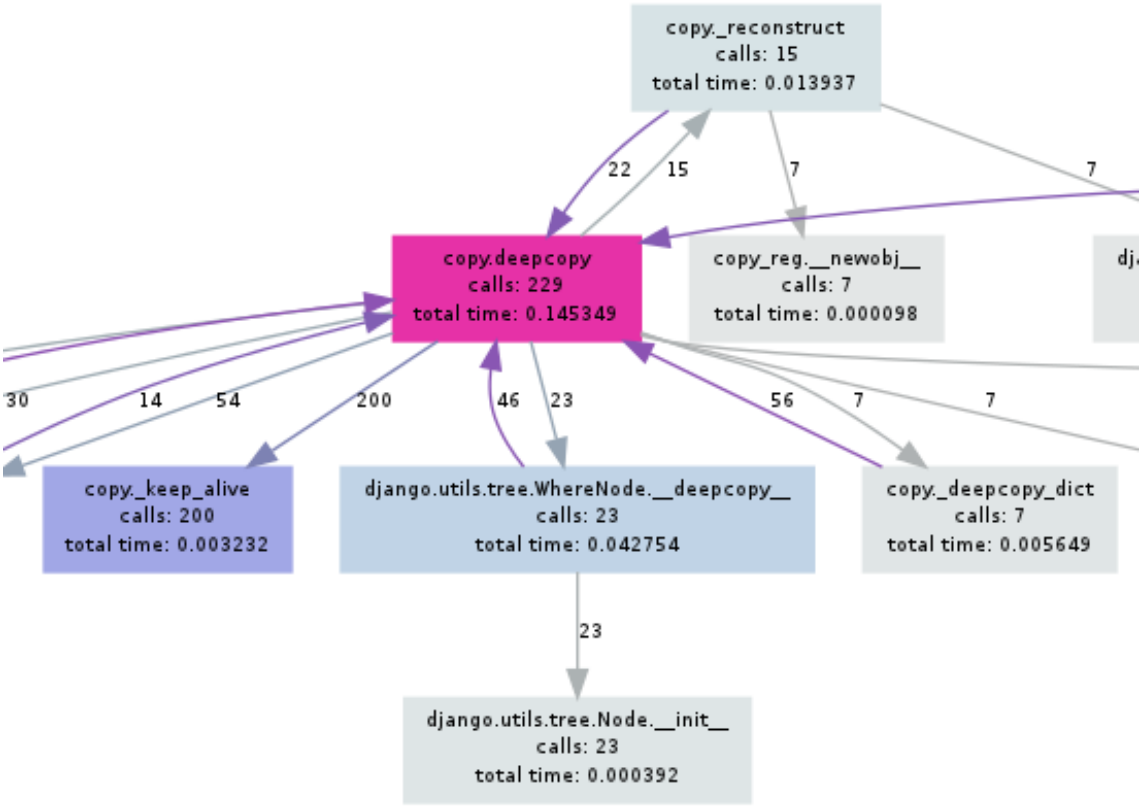
Činnost tohoto kódu je taková, že odešle požadavek na neexistující pohled jménem `nonexistent`. Tento krok by měl vyvolat výjimku. Pokud ji test zachytí, neodhalil chybu.

6.2 Profilování a optimalizace

Google app engine poskytuje kvalitní nástroj pro prohlížení logů z běhu interpretu aplikace [8]. Z těchto logů lze vyčíst jak dlouho trvaly jednotlivé dotazy na databázi, popř. datový sklad. Datový sklad v němž jsou uloženy soubory uživatelů vykazuje vysokou, konstantní latenci pohybující se okolo 1300 milisekund. Je zřejmé, že takto vysoká latence je už patrná uživateli. Pro optimalizaci bylo zvoleno řešení využívající memcache pro uložení souborů, které se často načítají. Extrakce souboru z memcache serveru trvá v průměru 56 milisekund. Velikost souboru není pro trvání přenosu klíčová.

Po těchto prvních optimalizacích založených na analýze údajů uložených Google app enginem byla provedena další analýza. Prvním krokem bylo zaznamenávání času volání funkcí a metod. Z těchto časů a jejich posloupnosti lze odhalit doba, kterou interpret prováděl instrukce v jednotlivých funkcích a metodách. Za pomocí modulu pycallgraph byl z těchto dat vygenerován graf (viz obr. 6.1) znázorňující počty volání funkcí a metod, původ těchto volání a doba vyhodnocení výsledku uvnitř funkce či metody.

Výsledky této analýzy byly v rozporu s původními předpoklady. Aplikace na mnoha místech využívá regulárních výrazů pro práci s řetězci. Ačkoli to bylo očekáváno, modul regulárních výrazů nebyl úzkým hrdlem aplikace. Dokonce se jednalo o jeden z neúspornějších modulů. Monitorovaný dotaz trval vyhodnotit přibližně 160 milisekund (viz obr. 6.1). Asi 145 milisekund prováděla aplikace kopírování dat pomocí funkce deepcopy. Deepcopy spouštěl modul překladu databázových dotazů z tečkové notace na SQL. Prováděl to vždy, když došlo k pokusu o provedení dotazu na databázi skrze metody objektu nebo třídy modelu. Po vysledování tras volání byl učiněn závěr, že jediný způsob jak tuto část aplikace zrychlit bylo omezit počet výše zmíněných překladů. Tohoto bylo docíleno sdílením mnoha informací mezi jednotlivými částmi aplikace tak, aby nemuselo docházet k opětovnému hledání téže informace.



Obrázek 6.1: Část z grafu volání funkcí a metod

6.3 Systémy, které vyvinutou aplikaci využívají

Aplikace je přístupná na http://www.herobe.com.

6.3.1 Hudební skupina Blues Bazaar

Hudební skupina Blues Bazaar je podle jejich manažera určena pro širší a žánrově rozmanitější spektrum blues-rockové hudby. Společně s kapelou Eric Clapton cover band měli svou webovou prezentaci umístěnou na webhostingu, který byl v minulosti provozován na doméně flyer.cz. 17. dubna 2013 neočekávaně ukončil svou činnost. V této chvíli má tato kapela svou webovou prezentaci umístěnou na bluesbazaar.herobe.com.

6.3.2 Hudební skupina Eric Clapton cover band

Kapela Eric Clapton cover band se k webové prezentaci na herobe.com dostala totožnou cestou jako hudební skupina Blues Bazaar. Jak její název napovídá, jedná se o hudební skupinu hrající skladby Erica Claptona. Její webová prezentace je umístěna na ericclapton-coverband.herobe.com .

6.3.3 WEB Project, s.r.o.

Tato společnost se zabývá vývojem informačních systémů, v této chvíli její vedení uvažuje o spolupráci s tímto projektem.

6.3.4 Cecilka benefits s.r.o. a Icarus s.r.o.

Obě firmy jsou uvedeny vedle sebe, neboť mají stejného majitele a tím je Tomáš Smetka. Pokud bude tento projekt pokračovat v nějaké komerční formě, stane se tak pravděpodobně skrze jednu z těchto společností. Zmíněné společnosti se zabývají tvorbou informačních systémů na klíč a dalšími doplňkovými službami jako vytvoření corporate identity, reklamních materiálů a tak podobně.

Kapitola 7

Závěr

Cílem této práce bylo vyvinutí systému pro sběr a prezentaci dat. Takový systém byl implementován pomocí webové služby. Skrze tu může uživatel navrhnout webovou aplikaci v prostředí, které má WYSIWYG podobu, ale dává uživateli i možnost vkládat konstrukce skriptovacího jazyka, popisující aplikační logiku. Důraz byl kladen i na možnost vkládat data do systému automaticky ze vzdáleného zdroje. Výsledná služba tak kombinuje vlastnosti rychlého vývoje ve wysiwyg grafickém nástroji se sílou snadno rozšiřitelného textového programovacího jazyka a umožňuje vytvořit téměř libovolný informační systém.

Zajímavostí vyvinuté aplikace je i vybraná platforma. Prostředí Google app engine je v mnohých ohledech nevšední, neboť se jedná o jednu ze služeb typu Paas (Platform as a service). V tomto prostředí je celá řada detailů infrastruktury skryta před vývojářem, což v kombinaci s vysokým stupněm virtualizace umožňuje efektivně spravovat zdroje. Vzhledem k rozvíjejícímu se trhu s řešeními v cloudu, lze usuzovat, že tato platforma je velmi perspektivní [8].

Aplikace s pracovním názvem Ilumexis, která byla v rámci této práce vyvinuta je v současnosti přístupná v provozu-schopném stavu na URL `http://www.herobe.com`, kde po dobu trvání beta verze je registrace a následné užívání nezpoplatněno. V případě budoucího rozšíření a zájmu trhu bude tato aplikace rozvinuta do podoby částečně zpoplatněné služby.

Aplikace v současné verzi umožňuje běh na subdoméně pod herobe.com se všemi vyvinutými moduly umožňující generování grafů, ochranu proti CSRF, komunikaci pře XML-RPC rozhraní a tak podobně. Zároveň umožňuje správcům jednotlivých systémů běžících na výše zmíněných subdoménách využívat automatické škálování zdrojů aplikačních serverů poskytované přímo službou Google app engine [8].

Z pohledu dalšího vývoje je aplikace připravena pro výrazné rozšíření integrovaného skriptovacího jazyka a WYSIWYG rozhraní, které využívá knihovnu TinyMCE. Pro tuto knihovnu existuje celá řada modulů, umožňujících například spravovat soubory nahrané uživateli na server. Z pohledu správců systému by bylo užitečné rozhraní protokolu FTP (File transfer protocol) nebo FISH (Files transferred over Shell protocol). Dále je možné využít kterékoliv rozšíření webového rámce Django, kterých je k dispozici značné množství.

Jak lze vidět, možností rozšíření je mnoho. V případě, že by se v jeho vývoji pokračovalo, by se v brzké době mohl z tohoto nástroje stát velice kvalitní pomocník pro tvorbu různých druhů webových aplikací.

Literatura

[1] Berners-Lee, T.; Fielding, R.; Masinter, L.: Uniform Resource Identifier (URI): Generic Syntax. RFC 3986 (INTERNET STANDARD), Leden 2005.
URL http://www.ietf.org/rfc/rfc3986.txt

[2] Fielding, R.; Gettys, J.; Mogul, J.; aj.: Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), Červen 1999, updated by RFCs 2817, 5785, 6266, 6585.
URL http://www.ietf.org/rfc/rfc2616.txt

[3] Holovaty, A.; Kaplan-Moss, J.: *The definitive guide to Django: Web development done right. 2. vyd.* New York: Springer-Verlag, 2009, iSBN 978-1-4302-1937-8.

[4] Joneja, N.: Google Cloud SQL: your database in the cloud. 2011-10-6 [cit. 14. května 2013].
URL http://googlecode.blogspot.cz/2011/10/google-cloud-sql-your-database-in-cloud.html

[5] Postel, J.: Transmission Control Protocol. RFC 793 (INTERNET STANDARD), Září 1981, updated by RFCs 1122, 3168, 6093, 6528.
URL http://www.ietf.org/rfc/rfc793.txt

[6] Rescorla, E.: HTTP Over TLS. RFC 2818 (Informational), Květen 2000, updated by RFC 5785.
URL http://www.ietf.org/rfc/rfc2818.txt

[7] Richardson, L.; Ruby, S.: *RESTful Web Services*. O'Reilly, 2007, iSBN 978-0-596-52926-0.

[8] Sanderson, D.: *Programming Google App Engine: Build and Run Scalable Web Apps on Google's Infrastructure*. O'Reilly Media, 2009, iSBN 978-0-596-52272-8.

[9] Shiflett, C.: Cross-Site Request Forgeries. 2004-12-13 [cit. 14. května 2013].
URL http://shiflett.org/articles/cross-site-request-forgeries

[10] Sterne, B.: Origin Header Proposal. 2011 [cit. 14. května 2013].
URL http://people.mozilla.org/~bsterne/content-security-policy/origin-header-proposal.html

[11] VAN ROSSUM, G.: Python - zdrojové kódy [online]. [cit. 2012-12-18].
URL http://svn.python.org/view/*checkout*/python/trunk/Misc/HISTORY

[12] WWW stránky: Django 1.4 documentation [online]. 2008-11-01 [cit. 14. května 2013].
URL https://docs.djangoproject.com/en/1.4/

[13] WWW stránky: Cross-Site Request Forgery (CSRF). 2013-9-1 [cit. 14. května 2013].
URL
https://www.owasp.org/index.php/Cross-Site_Request_Forgery_%28CSRF%29

[14] WWW stránky: Django 1.2.5 release notes. [cit. 14. května 2013].
URL https://docs.djangoproject.com/en/1.2/releases/1.2.5/

[15] WWW stránky: Google Cloud Storage introduction. [cit. 14. května 2013].
URL https://developers.google.com/storage/docs/getting-started

[16] WWW stránky: SOAP Version 1.2 Part 1: Messaging Framework (Second Edition).
[cit. 14. května 2013].
URL http://www.w3.org/TR/2007/REC-soap12-part1-20070427/

Dodatek A

Obsah CD

Součástí této práce je CD, které obsahuje:

- Zdrojové kódy implementované aplikace
- Elektronická verze této technické zprávy ve formátu PDF
- Zdrojové kódy technické zprávy ve formátu LaTeX

Dodatek B

Pokyny pro instalaci

Aplikace je implementována v jazyce Python 2.7 a i když přímo pro běh aplikace na vzdáleném serveru není instalace interpretu tohoto jazyka zapotřebí, mnoho nástrojů potřebných pro jeho správu je vyvinutých také v tomto jazyce. Ve většině distribucí GNU/Linuxu je interpret tohoto již přítomen. Pokud tomu tak není, nainstalujte jen přes balíčkovací systém Vaší distribuce. V případě, že používáte jiný operační systém jež GNU/Linux, postupujte při jeho instalaci podle instrukcí na http://www.python.org.

Kroky instalace:

1. Registrace na Google app engine

   Na adrese https://appengine.google.com/ se registrujte nebo použijte již existující Google účet. Založte novou aplikaci s unikátním názvem. Ten je užit pouze interně. Vaše aplikace od té chvíle poběží na adrese `VAŠE_APLIKACE.appspot.cz`.

2. Instalace Google App Engine SDK for Python

   Na adrese https://developers.google.com/appengine/ naleznete ke stažení Google App Engine SDK for Python. Tento software stáhněte a nainstalujte.

3. Instalace a konfigurace databázového prostředí

   Po přihlášení na Google účet aktivujte přístup ke konzoli Google API na adrese `https://code.google.com/apis/console/`. Pod záložkou Services aktivujte položku Google Cloud SQL a Google Cloud Storage. Tento krok může vyžadovat zadání údajů pro fakturaci. V hlavním menu se Vám zobrazí aktivovaná API. V sekci Google Cloud SQL vytvořte instanci databázového systému. Přidejte do seznamu autorizovaných aplikací název aplikace běžící na Google app engine.

4. Instalace a konfigurace Google cloud storage

   V konzoli Google API (https://code.google.com/apis/console/) přejděte do sekce Google Cloud Storage. Přejděte do podsekce s názvem Interoperable Access. Tuto funkcionalitu aktivujte. Zaznamenejte si zobrazené hodnoty Access Key a Secret (toto pole se zobrazí po kliknutí na tlačítko secret).

5. Instalace webového rámce Django

   Ze stránky `http://www.djangoproject.cz/download/` stáhněte soubor Django-1.4.5.tar.gz a po rozbalení nainstalujte spuštěním skriptu `setup.py`. Lze stáhnout i novější verzi, ačkoli u ní již není kompatibilita zaručena, neboť aplikace byla testována na verzi 1.4.

6. Vytvoření nové instance aplikace

   Z přiloženého CD zkopírujte adresář Ilumexis do libovolného umístění na Vašem lokálním diskovém oddílu.

7. Registrace domény

   Zaregistrujte si doménu u libovolného registrátora.

8. Konfigurace aplikace

   V adresáři, který jste vytvořili v předchozím kroku naleznete soubor `app.yaml`. Otevřete jej v textovém editoru (Gedit, Kate, Poznámkový blok apod.). Na prvním řádku naleznete `application: ilumexis`. Zaměňte `ilumexis` za jméno aplikace zvolené v prvním kroku.

   Společně se souborem `app.yaml` by ve stejném adresáři měl být přítomen podadresář `ilumexis`. V něm naleznete soubor `settings.py`. Ten otevřete v textovém editoru (Gedit, Kate, Poznámkový blok apod.). Do příslušných hodnot vložte:

   - TLD
     Například pokud by Vaše registrovaná doména byla `herobe.com`, hodnota by byla `com`.
   - SLD
     Například pokud by Vaše registrovaná doména byla `herobe.com`, hodnota by byla `herobe`.
   - Administrátory
     N-tice dvojic jména a emailu administrátorů.
   - Google storage access credentials
     Tyto konstanty odpovídají hodnotám Access Key a Secret z kroku 4.
   - Prefix názvů bucket entit v Google cloud storage
     Libovolný delší řetězec s pokud možno vysokou entropií.
   - Databáze
     Přístupové údaje k databázi Google cloud SQL či libovolné jiné.

   Zbylé hodnoty můžete rovněž upravit. Jejich nastavení již není kritické pro činnost systému.

9. Konfigurace DNS záznamů

   V DNS administraci u Vašeho registrátora vytvořte záznam typu `CNAME`, který bude odkazovat na `ghs.googlehosted.com`.

10. Instalace portálu

    Aby byl systém provozuschopný, musí být nainstalovaný hlavní systém pojmenovaný `www`. Bez něm nelze instalovat další systémy. Tento systém se musí nainstalovat ručně provedením speciálních SQL dotazů v konzoli Google cloud SQL. Tyto SQL dotazy naleznete v `portal.sql` ve stejném adresáři jako `app.yaml`.

11. Uvedení aplikace do provozu

    Po provedení předchozího kroku byste měli mít přístup do administrace hlavního systému pod uživatelským jménem `root` a heslem root. Toto heslo je silně doporučeno ihned změnit.