VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

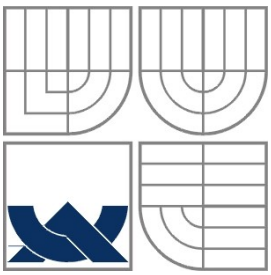
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ŘÍDICÍ SYSTÉM GENERÁTORU VĚDECKÝCH WEBOVÝCH PORTÁLŮ

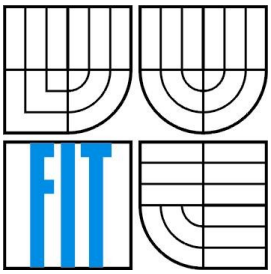
BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE JAN HORÁK
AUTHOR

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ŘÍDICÍ SYSTÉM GENERÁTORU VĚDECKÝCH WEBOVÝCH PORTÁLŮ

MANAGEMENT SYSTEM OF THE SCIENTIFIC WEB PORTAL GENERATOR

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR

JAN HORÁK

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. RNDr. PAVEL SMRŽ, Ph.D.

BRNO 2008

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2007/2008

Zadání bakalářské práce

Řešitel: **Horák Jan**
Obor: Informační technologie
Téma: **Řídící systém generátoru vědeckých webových portálů**
Kategorie: Web

Pokyny:

1. Seznamte se s XML-RPC, případně dalšími relevantními standardy pro řízení distribuovaných výpočtů.
2. Vytvořte řídící část systému, který umožní uživateli kontrolovat běžící procesy, určovat jejich priority atd.
3. Navrhněte a implementujte rozhraní pro informační službu, která bude prostřednictvím různých protokolů informovat uživatele o stavu zpracování zadané úlohy.
4. Vyhodnoťte vytvořený systém pomocí standardních metrik.
5. Vytvořte stručný plakát prezentující práci, její cíle a výsledky.

Literatura:

- podle dohody

Při obhajobě semestrální části projektu je požadováno:

- funkční prototyp řešení

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Smrž Pavel, doc. RNDr., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2007
Datum odevzdání: 14. května 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Pavel Zemčík
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. **Pan**

Jméno a příjmení: **Jan Horák**
Id studenta: 79350
Bytem: Jílová 1483, 583 01 Chotěboř
Narozen: 24. 01. 1986, Hradec Králové
(dále jen "autor")

a

2. **Vysoké učení technické v Brně**

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
bakalářská práce

Název VŠKP: Ruby on Rails pro generátor vědeckých webových portálů
Vedoucí/školitel VŠKP: Smrž Pavel, doc. RNDr., Ph.D.
Ústav: Ústav počítačové grafiky a multimédií
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:
tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☐ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

.....	.....
Nabyvatel	Autor

Abstrakt

Práce analyzuje způsoby řízení distribuovaných výpočtů a popisuje analýzu, návrh a implementaci Řídicího systému generátoru vědeckých webových portálů. Cílem práce bylo propojit ostatní součásti aplikace, umožnit uživatelům kontrolovat dlouhotrvající výpočty a měnit jejich priority.

Klíčová slova

web, portál, python, xml-rpc, jabber

Abstract

This BSc Thesis analyses possible ways to control distributed computations, describes analysis, conceptual design and implementation of the Management system of the scientific web portal generator. The management system connects other parts of application, allows users to control long-running computations and to change its priorities.

Keywords

web, portal, python, xml-rpc, jabber

Citace

Jan Horák: Řídicí systém generátoru vědeckých webových portálů, bakalářská práce, Brno, FIT VUT v Brně, 2008

Řídicí systém generátoru vědeckých webových portálů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením doc. RNDr. Pavla Smrže, Ph. D.
Další informace mi poskytl Marek Schmidt, Bc.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jan Horák
11.5.2008

Poděkování

Rád bych poděkoval vedoucímu bakalářské práce doc. RNDr. Pavlu Smržovi, Ph. D za vstřícný přístup a hodnotné připomínky při vedení bakalářské práce.
Zároveň bych rád poděkoval Marku Schmidtovi, Bc. za ochotnou pomoc při řešení problémů spojených s využitím jazyka Python pro generování webových stránek.

Obsah

1 Úvod

Cílem bakalářské práce bylo navrhnout a implementovat Řídicí systém generátoru vědeckých webových portálů. Projekt jako celek rozhodně přesahuje rozsah jedné bakalářské nebo diplomové práce, proto se na jeho tvorbě podílelo současně více studentů, zpravidla v rámci svých bakalářských a diplomových prací, přičemž řídící systém tyto části spojuje.

Projekt si klade za cíl pomocí vyhledávacích služeb na internetu získat a archivovat informace o osobách z vědecké oblasti a jejich dílech. Hledají a analyzují se vztahy mezi nimi a přehlednou formou bude v budoucnu tyto získané informace prezentovat.

Kapitola Analýza řídicího systému vysvětluje některé důležité pojmy, popisuje použité nástroje, základní rozdělení generátoru vědeckých webových portálů na jednotlivé části a nastíní základní funkci řídicího systému. Popsány budou i problémy, na které bylo nutné se zaměřit při následném návrhu a při implementaci.

Následuje kapitola Návrh Řídicího systému, která již konkrétně popisuje zvolené metody a detailněji rozebírá návrh celého Řídicího systému a jeho rozdělení na jádro a uživatelské rozhraní.

Kapitola Implementace Řídícího systému přiblíží detaily samotné programové části a rozebrány budou použité nástroje a standardní i nestandardní řešení, která v práci byla použita. Jsou zde popsány i některé problémy, se kterými se autor v průběhu implementace setkal, a jejich řešení.

V kapitole Závěr jsou zhodnoceny dosažené výsledky a nastíněno je i další možné pokračování vývoje Řídicího systému, stejně jako celého projektu.

2 Analýza Řídicího systému

V této kapitole je popsána základní struktura celého systému. Rozebrány budou použité metody a nástroje a ve stručnosti budou popsány i jednotlivé části Řídicího systému, jejich funkce a cíle.

2.1 Použité pojmy

Zde jsou ve stručnosti popsány základní pojmy, protokoly a služby, kterých se bude v práci dále využívat.

2.1.1 Distribuovaný výpočet

Jedná se o způsob provádění jisté netriviální úlohy, kdy se úloha jako celek rozdělí na menší části, které mohou být vykonávány paralelně nebo dokonce na různých strojích. Vstupy a výstupy jednotlivých dílčích částí putují mezi počítači zpravidla přes síť. Správné a efektivní nastavení všech parametrů distribuovaného systému je v mnoha případech velmi obtížné.

2.1.2 RPC

Zkratka anglického názvu *Remote procedure call* (česky vzdálené volání procedur) je v mnoha distribuovaných výpočtech základním prvkem komunikace mezi jednotlivými částmi systému. Tato technologie dovoluje jednomu programu zavolat jistou metodu na jiném počítači, jako by to byla metoda lokální. Počítač, který je tímto protokolem volán, vrátí výsledek výpočtu zpět volajícímu.

2.1.3 XML

Zkratka znamená *eXtensible Markup Language* (česky rozšiřitelný značkovací jazyk). Jedná se o zobecněný značkovací jazyk, standardizovaný konsorciem W3C. Nabízí velmi široké využití od definice dokumentů po komunikaci a výměnu dat mezi aplikacemi. K dokumentu se může připojit vzhled, pomocí kterého lze dokument prezentovat.

2.1.4 XML-RPC

XML-RPC je protokol, který vznikl spojením značkovacího jazyka XML a technologie pro vzdálené volání procedur RPC. Požadavky na vzdálené volání procedur jsou zasílány ve formě XML zpráv, stejně tak jejich odpovědi.

Tuto technologii lze využít v mnoha jazycích, neboť protokol XML-RPC podporují knihovny většiny běžně používaných programovacích jazyků. Zprávy jsou potom sestavovány nezávisle

na programovacím jazyku, a proto není problém jejich současná kombinace (jiným způsobem obtížně proveditelná). S tím ale souvisí některá omezení, zejména neobvyklých datových typů, které jsou pro daný jazyk určitým způsobem specifické. Takové konstrukce v jiném jazyku nemusí existovat a protokol XML-RPC takové typy tím pádem vůbec nepodporuje.

2.1.5 Jabber

Jabber je komunikační protokol se základem v jazyce XML. Slouží pro komunikaci mezi uživateli z celého světa (tzv. Instant messaging). Oproti masivně rozšířenému protokolu ICQ je jeho výhodou především otevřený protokol a tím pádem snadnější implementace i dostupnější podpora pro vývoj aplikací nad tímto protokolem.

Uživatel je jednoznačně identifikován pomocí tzv. JID, neboli identifikátoru Jabber ID. To se skládá z uživatelského jména, jména serveru, na kterém je uživatel přihlášen, a tzv. zdroje (resource). Zdroje umožňují vícenásobné připojení jednoho uživatelského účtu na jednom serveru v jeden časový okamžik [7].

2.1.6 SQLite3

Jedná se o jednoduchý relační databázový systém. Databáze není na rozdíl od jiných běžně využívaných databázových systémů k dispozici jako služba běžícího serveru, ale je celá uložena v jednom souboru. Pro přístup k datům tak není potřeba instalovat žádný jiný software nebo databázový server.

Práce s databází probíhá obdobně, jako s jinými relačními databázemi, pomocí příkazů jazyka SQL, přičemž systém SQLite3 obsahuje téměř kompletní specifikaci SQL92. Výkonnostní parametry databázového systému SQLite3 sice nejsou tak dobré, jako má například běžný MySQL server, ale pro potřeby Řídicího systému budou zajisté stačit. Rozdíly ve výkonnosti oproti MySQL jsou totiž patrné až při větším zatížení, naopak oproti databázovému systému PostgreSQL prokazuje SQLite3 v mnoha rychlostních testech výsledky o mnoho lepší [15].

Zajímavá je i práce s datovými typy. Ty mají totiž oproti ostatním systémům (např. MySQL) vesměs pouze doporučující charakter. To znamená, že do sloupce typu INTEGER lze uložit jak celé číslo, tak i řetězec nebo číslo v plovoucí řádové čárce. Data ukládaná do databáze by se tedy měla kontrolovat explicitně, zda odpovídají požadovanému datovému typu, jinak se do databáze dostanou nekonzistentní data, a to je v každém případě nesprávné.

Na druhou stranu SQLite3 kontroluje unikátnost primárního klíče, který lze navíc generovat automaticky. V tomto ohledu tedy není nutné provádět žádná speciální ošetření.

2.1.7 SQL injection

Jedná se o jeden ze základních útoků na webové aplikace, konkrétně na jejich databázi. V případě, že se útočníkovi (v horším případě i náhodou běžnému uživateli) podaří do SQL dotazu vložit takovou sekvenci znaků, která ovlivní chování a výsledek SQL dotazu, jedná se o tzv. SQL injection.

Tímto způsobem útočník může v lepším případě shodit server, v horším případě se dokonce dostat k citlivým datům ostatních uživatelů, případně k administrátorským právům. Proto je nezbytně nutné možnosti takových útoků eliminovat.

Základní ochranou je ošetření dat vkládaných do SQL dotazu od nebezpečných znaků (zejména apostrofy). Ještě bezpečnější je využití parametrizovaných dotazů.

2.1.8 Mod python

Tato knihovna je volitelnou součástí webového serveru apache. Jedná se o prostředníka mezi webovým serverem a interpretem jazyka Python na daném serveru. Modul mod_python nabízí pro zobrazování stránek tři základní možnosti (tzv. Handlery) – PSP, Publisher a CGI.

PSP Handler je velmi podobný vkládanému skriptu jazyka PHP. Jedná se tedy o skript, který se má vykonat na straně serveru, a který je vkládán přímo do HTML kódu stránky pomocí speciálních tagů. Toto řešení je vhodné pro menší, dynamické stránky, neboť kombinace HTML a vkládaného skriptu by ve větším projektu s rostoucím rozsahem ztratilo na přehlednosti.

CGI Handler je rozhraní pro přístup ke službám serveru pomocí CGI (Common Gateway Interface).

Publisher Handler je modul vhodný pro složitější, komplexnější projekty. Je inspirován modulem Zpublisher ze systému Zope (rozsáhlý systém pro správu obsahu a pokročilé generování stránek). Tento modul dovoluje striktně oddělit logiku od generování HTML kódu, což je u větších projektů nezbytné. I díky této vlastnosti byl právě tento modul využit v Řídicím systému.

2.2 Struktura systému

Jak již bylo zmíněno v úvodu, byl projekt kvůli své rozsáhlosti rozdělen na několik samostatných částí (modulů), na kterých se jednotliví členové týmu podílejí samostatně. Řídicí systém má na starost tyto dílčí části propojit a umožnit jejich vzájemnou komunikaci.

Řídicí systém je z velké části autonomní, ale některé úkony musí provést uživatel sám. Proto je součástí Řídicího systému i uživatelské rozhraní, pomocí kterého jsou úlohy do systému zadávány a kontrolovány. Toto rozhraní slouží zároveň i jako rozhraní pro administrátora, který pomocí něho může upravovat vlastnosti systému.

Rozhraní je implementováno jako webová aplikace, s nutností autentifikace a podporou odlišných práv pro běžné uživatele a pro administrátora.

2.3 Volba implementačního jazyka

Implementačním jazykem byl zvolen Python, neboť jde o moderní nástroj vhodný pro vývoj jak webových aplikací, tak běžných konzolových aplikací, běžících například na serveru jako síťová služba. Jeho výhodou je pokročilá podpora objektového návrhu, jasná a čitelná syntaxe, rozsáhlá podpora základních knihoven a modulů a v neposlední řadě i velké množství aplikací a knihoven třetích stran, které jsou velmi často k dispozici jako otevřený software (Open Source).

Díky zmíněným výhodám lze mnoho problémů implementovat o poznání rychleji, než by to bylo možné v jiných jazycích, například v nízkoúrovňovém jazyku C. Pokud se má užitím jednoho jazyka implementovat jak webová, tak běžná konzolová aplikace, přichází jako alternativa v úvahu ještě jazyk Ruby a framework Ruby on Rails, ve kterém jsou psány některé jiné moduly Generátoru vědeckých portálů.

2.4 Volba protokolu pro distribuovaný výpočet

Existuje několik protokolů pro distribuované výpočty, jedněmi z nejznámějších jsou *XML-RPC*, *WDDX* (Web Distributed Data eXchange) *a SOAP* (Simple Object Access Protocol). Posledně jmenovaný je vyvíjen pod záštitou konsorcia W3C, vychází částečně z XML-RPC a částečně z WDDX a má být do budoucna nástupcem právě XML-RPC.

Při výběru vhodného protokolu pro generátor vědeckých webových portálů byl kladen důraz na jeho podporu v různých jazycích a s tím související i míra rozšíření protokolu. V tomto ohledu byl nejvhodnější variantou protokol XML-RPC, který je sice starší než například SOAP, ale jeho podpora napříč jazyky je pravděpodobně nejlepší ze všech tří jmenovaných.

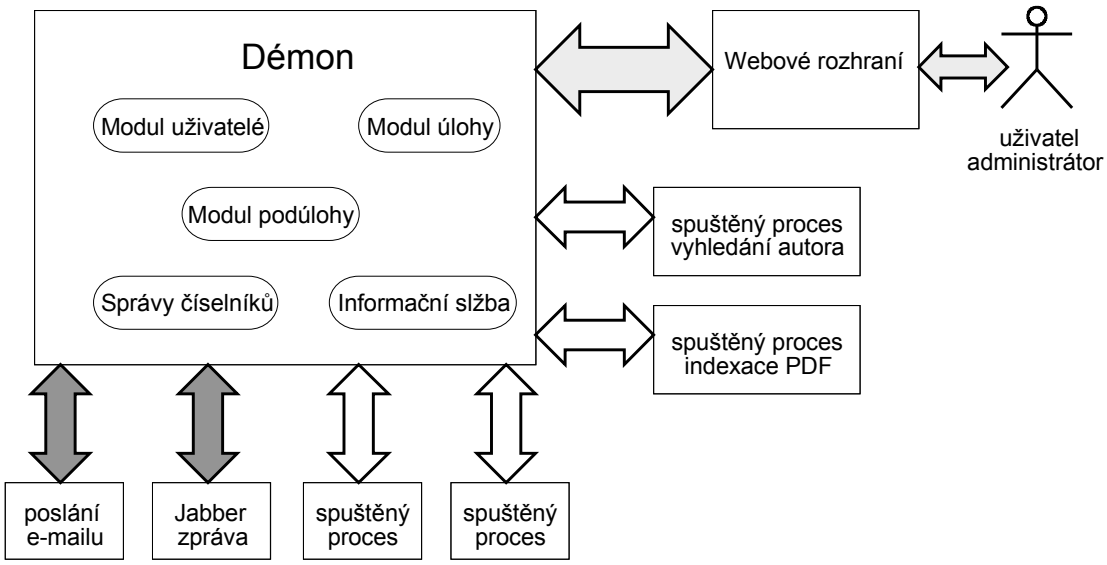
2.5 Rozdělení Řídicího systému

Ve svých počátcích analýza Řídicího systému obsahovala jeden centrální objekt, který byl pracovně nazván *Démon*. Ten měl obsahovat jak logickou část, která by řídila ostatní procesy, tak webové rozhraní, kterým uživatelé definují vlastnosti úloh, procesů, jejich priority apod. Znamenalo by to prakticky to, že webová aplikace by jako jádro řídila celý systém.

Kombinace řídicí části a webové služby do jedné aplikace se však ukázala jako nevyhovující, takže bylo nutné zcela oddělit logiku Řídícího systému (démona) od webového rozhraní. Webové

rozhraní se tak prakticky stalo jednou ze součástí generátoru – samostatným modulem, který komunikuje s Řídicím systémem stejně, jako například modul pro indexaci PDF.

Rozdělení Řídicího systému a náznak komunikace mezi jednotlivými částmi je uvedeno na obrázku č. 1.



Obrázek 1: Schéma rozdělení Generátoru vědeckých webových portálů a naznačení komunikace mezi jednotlivými moduly

Aby byl zřejmý rozdíl mezi *modulem* a *procesem*, následuje krátké vysvětlení: Modulem rozumíme aplikaci (jednu ze součástí Generátoru) a procesem potom spuštění této aplikace. Proces tedy vždy odpovídá nějakému modulu, ale má navíc definovaný vstup. Může tedy v jeden okamžik existovat mnoho procesů odpovídající stejnému modulu, ne naopak. Proces ve smyslu přiřazení k určité úloze může být zároveň nazýván podúlohou (subtask).

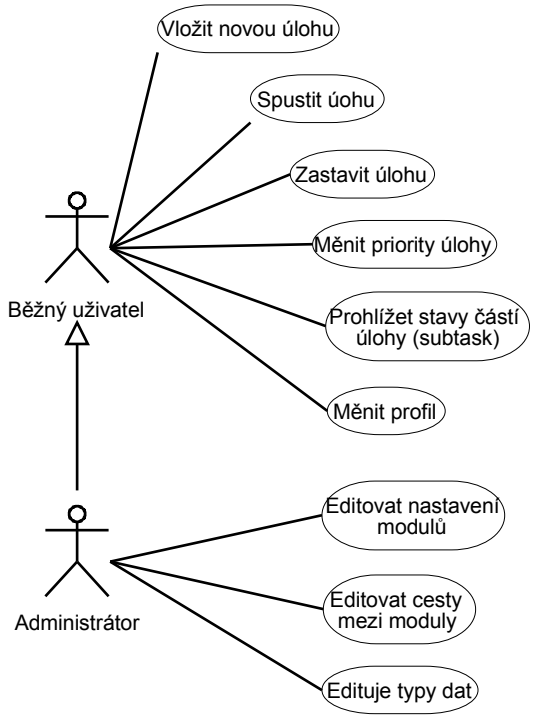
2.6 Uživatelské webové rozhraní

Uživatelské rozhraní pro ovládání celého systému je implementováno jako víceuživatelská webová služba. Přes toto webové rozhraní se do Řídicího systému zadávají nové úlohy, kontrolují se běžící úlohy, spravují se jejich atributy a je možné zobrazit i dokončené úlohy. Webové rozhraní slouží jako editační a informační centrum jak pro běžného uživatele, tak pro administrátora Řídicího systému. Proto musí nutně podporovat autentizaci, přičemž se rozlišují tyto dvě základní role: *běžný uživatel* a *administrátor*.

Běžný uživatel má právo přidat do Řídicího systému novou úlohu, své úlohy může spustit, pozastavit nebo zrušit. Jednotlivým úlohám může měnit priority, podúlohám (jakožto částem úlohy) potom může měnit pořadí. Běžný uživatel smí dále editovat svůj profil (údaje jako e-mail, osobní stránky atd.) a změnit si heslo pro přihlášení.

Administrátor má stejná práva jako běžný uživatel a k tomu některá další. Může přidávat nové uživatele, jejichž účty může editovat, aktivovat i deaktivovat. Administrátor má navíc práva zobrazovat a editovat úlohy (Tasks) všech uživatelů.

Uživatelské webové rozhraní slouží i k administraci celého systému, např. přidávání nových modulů, editují se zde propojení mezi moduly a řada dalších možností a nastavení.



Obrázek 2: Případ užití webového rozhraní

2.7 Rozdělení webového generátoru na moduly

Jak již bylo zmíněno, generátor webových portálů bude obsahovat několik modulů (jednotlivé části implementované odděleně od Řídicího systému), které budou autonomně provádět jednoznačně definované úkony a ze kterých se bude celý generátor skládat. Tyto moduly bude Řídicí systém spouštět, předávat jim vstupy a přebírat od nich výstupy.

2.7.1 Seznam modulů:

Zde je stručný seznam modulů, které by měly být zařazeny do generátoru:

- Uživatelské rozhraní Řídicího systému
- Rozpoznávání definovaných entit na webové stránce
- Manuální úprava nalezených entit uživatelem
- Vyhledání osobních stránek autorů publikací
- Indexace nalezených PDF - publikací
- Pokročilá prezentace získaných dat

Celý proces začíná definicí URL vstupní stránky na internetu, kde se nachází seznam vědeckých pracovníků, resp. zadáním seznamu jmen přímo. Systém se snaží rozpoznat vyjmenované entity na zadané webové stránce, tyto analyzuje a po skončení analýzy odevzdá ve strukturované formě, ve formátu XML.

Následuje ruční úprava výsledků automatického rozpoznání vyjmenovaných entit v případě, že je výsledek nepřesný nebo si to uživatel explicitně vynutí. Tento krok není potřeba provádět vždy.

Další součástí, na které částečně stojí i úspěšnost celého systému, je vyhledání osobních stránek zadaných osob podle vyhledaných nebo zadaných entit pomocí služeb internetových vyhledávačů. Na domovské stránce dané osoby se pokusí nalézt stránku s publikacemi a na ní se pokusí identifikovat odkazy na publikace samotné.

Pokud jsou nalezeny publikace ve formátu PDF, jsou staženy a další modul tyto soubory indexuje.

Nakonec budou výsledky archivovány a posléze přehlednou formou zobrazovány a publikovány. Tato část systému není prozatím implementována a pravděpodobně bude oddělena od samotného generátoru.

3 Návrh Řídicího systému

V této kapitole je detailněji rozebrán návrh Řídicího systému a diskutovány jsou řešení některých problémů, které během návrhu nastaly.

3.1 Jádro Řídicího systému

Řídicí systém byl navržen tak, že existuje jeden centrální prvek (arbitr), který je označován jako *Démon*. Démon proto, protože na serveru běží v nekonečné smyčce jako služba. Konkrétně jde o upravený XML-RPC server, který na vyhrazeném portu poskytuje služby RPC (vzdáleného volání procedur).

Démon čeká na požadavky od klientů, kterými jsou v tomto případě úlohy rozdělené na podúlohy (procesy). Tím, jak reaguje na tyto požadavky a jak vytváří nové podúlohy, řídí průběh celého výpočtu.

3.2 Webové uživatelské rozhraní

Jak bylo zmíněno výše, uživatelské rozhraní je vlastně webová služba, psaná v jazyce Python, která s Řídicím systémem komunikuje stejně, jako ostatní součásti generátoru – pomocí protokolu XML-RPC.

3.3 Provádění procesů

Pro ilustraci, jak bude nějaký proces provádět svůj úkol (tedy jednu konkrétní podúlohu – subtask), si vezměme například proces *vyhledání osobní stránky jistého autora*. Tento proces bude mít na vstupu například *jméno vědeckého pracovníka* a jako výstup nalezenou *adresu jeho osobních stránek*. Proces poběží jako samostatná aplikace, kterou bude posléze spouštět Řídicí systém, a s Řídicím systémem bude komunikovat přes XML-RPC protokol.

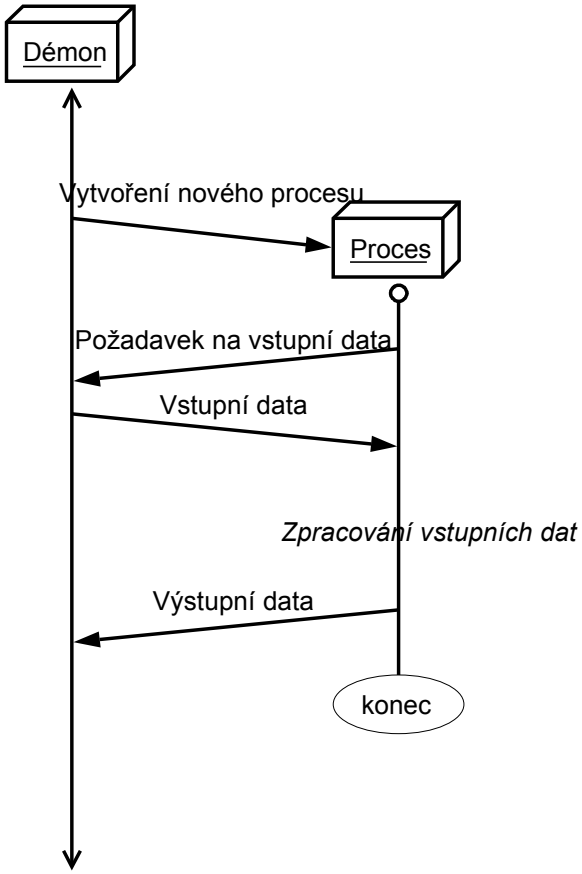
Dále je nutné spojit tento proces, se záznamem v Řídicím systému (konkrétně položka podúlohy). To je zařízeno předáním jednoznačného *ID* aplikaci, jako vstupního parametru. Toto ID je v procesu uchováno a použito vždy při komunikaci pomocí XML-RPC protokolu, aby Řídicí systém jednoznačně věděl, s kým komunikuje.

Ukázkový proces na počátku požádá Řídicí systém o vstupní data, která dostane ve formě XML dokumentu jako řetězce. Potom proces provede svojí činnost a získá nějaká výstupní data, která

opět zformátuje do podoby XML dokumentu. Tyto výstupní data předá zpět Řídicímu systému, který je uloží, zpracuje a v případě potřeby předá dál.

Formát výstupních dat, resp. *struktura XML dokumentu* s výstupem, určuje, co se bude s daty dále provádět, proto je nutné tento formát dodržovat. Kromě tohoto je výběr následující akce, resp. procesu, závislý na definovaných cestách, kterými jsou jednotlivé moduly propojeny. Více o propojování modulů v následující kapitole.

Zde je ukázka komunikace démona Řídicího systému s jednou z mnoha aplikací (procesem, podúlohou).



Obrázek 3: Ukázka komunikace Řídicího systému s procesem

3.4 Propojení modulů

Z počátku byl návrh takový, že způsob, jakým budou jednotlivé moduly propojené, bude definovaný staticky a výstup jednoho procesu půjde bez jakéhokoliv zásahu jako vstup dalšího procesu. Toto řešení se ovšem ukázalo jako nedostačující, protože takovéto neobecné řešení by si s sebou neslo mnoho problémů a omezení. Například by nebylo možné jednoduše přidat modul, pokud by již nějakým způsobem byly ostatní zadefinované.

V průběhu návrhu nebyly známé všechny detaily jednotlivých modulů, neboť byly také ve stádiu návrhu a implementace. A v neposlední řadě i absence možnosti rozdělit výstup jednoho procesu na několik vstupů dalších procesů vedlo definitivně ke změně návrhu na obecnější a dynamičtější způsob propojení jednotlivých modulů.

3.4.1 Dynamické propojení procesů

V jádru Řídicího systému jsou definovány moduly, které zastupují samostatně implementované aplikace – součásti generátoru. U těchto modulů se definuje zejména cesta pro spuštění.

Potom jsou definovány cesty mezi moduly, které kromě spojení dvou modulů definují i typ informace, který je touto cestou přenášen. Tento typ informace se posléze zohledňuje ve vstupech a zejména výstupech jednotlivých procesů. Pokud se totiž k Řídicímu systému dostane výsledek od jistého procesu, zjišťuje se právě podle typu informace (resp. názvem XML elementů), k jakému procesu se má dále pokračovat (resp. jaký modul se má použít).

Typy informací, které se mezi procesy šíří, jsou definovány jednoznačným alfanumerickým názvem, který bude posléze použit i jako název elementu v XML výstupních dat.

3.4.2 Dělení úlohy na podúlohy

Řešením dílčího problému tedy bude obecně vykonání určitého procesu (podúlohy) s definovanými daty. Několik takovýchto podúloh potom vytváří úlohu. Podúloha u sebe tedy uchovává informaci, ke které úloze patří a jaké informace zpracovává (tedy jaké cestě mezi moduly odpovídá). Dále uchovává i informaci, kolikrát už byly jisté informace zpracovávány od vytvoření první podúlohy. Při každém předání prvku dále se toto vnitřní počítadlo inkrementuje, aby se nepřekročila maximální hloubka zanoření. Pro pozdější analýzu jsou také důležité časy, kdy byla podúloha vytvořena, kdy spuštěna a kdy zastavena (příp. ukončena).

3.4.3 Paralelní zpracování podúloh

Jedním ze základních úkolů Řídicího systému bylo implementovat možnost paralelního zpracování více úloh, přičemž jednotlivé podúlohy jsou na sebe nějakým způsobem navázány. Jejich současný běh a koordinaci lze řešit více způsoby.

Tím nejprimitivnějším by bylo spouštět všechny podúlohy jakmile jsou známy jejich vstupy. Pokud by v takovém případě měl jeden modul na výstupu například 100 webových adres, které by se měly postupně zpracovat, vzniklo by tak v operačním systému najednou 100 nových procesů, běžících paralelně. Počet paralelně běžících procesů by tak lavinovitě narůstal, což je samozřejmě nepřijatelné. Ani jejich omezení na nějaký předem definovaný počet nic neřeší, naopak by se mohly zahazovat významné informace.

Koordinace paralelně běžících podúloh se tedy začala řešit frontou. Pro celou aplikaci lze potom nastavit, kolik procesů (podúloh) jednoho uživatele může běžet paralelně. Pokud je limit dosažen, čeká se na uvolnění prostředků, tedy podúloha čeká ve frontě.

Jakmile jeden proces skončí, Řídicí systém zkontroluje, zda ve frontě existují nějaké čekající podúlohy. Pokud ano, spustí se první z nich. Jejich pořadí je tedy důležité zachovat, aby nedošlo k uváznutí nějakých podúloh. Zároveň se při výběru úlohy bere v úvahu priorita celé úlohy, kdy prioritnější úlohy mají vždy přednost před méně prioritními.

3.5 Uchování dat v aplikaci

Původní uchování dat v aplikaci bylo velmi primitivní – jako datové struktury přímo v aplikaci. To ovšem absolutně nevyhovovalo požadavkům na objem dat, který bude v průběhu aplikace velmi rychle stoupat a brzy by mohlo dojít k vyčerpání paměťových možností. Bylo tedy zřejmé, že data v aplikaci se musí ukládat odděleně od aplikace.

Druhým návrhem, jak implementovat ukládání dat v aplikaci, bylo vytvořit jeden nebo více XML souborů. To by ovšem nutně znamenalo vytvořit nějakou vrstvu mezi daty aplikace a samotným XML a to by zabralo zbytečně mnoho času, přičemž ani použití by nebylo úplně pohodlné.

Již ze struktury dat a s přihlédnutím na nároky na jejich objem a jednoduchost přístupu se nabízela možnost využít pro uložení dat v aplikaci *databázi*. Instalovat kvůli tomu databázový server by ale nebylo nejelegantnější řešení, proto se nejlepším řešením ukázala knihovna SQLite3 (více o ní viz. kapitola 2.1.6).

3.5.1 Třída DB – vrstva mezi databází a moduly

Práce s databází SQLite se od jiných relačních databází v principu neliší. Knihovna Pythonu *sqlite3* nabízí několik standardních metod pro práci s SQL dotazy i pro zpracování jejich výsledků. Vzhledem k tomu, že např. pro vkládání a editaci jednotlivých řádků neexistují speciální metody, musí se tyto dotazy tvořit přímo v programu. Tento nízkoúrovňový přístup dává sice absolutní volnost, ale neustálá tvorba zpravidla podobných SQL dotazů není zajisté efektivní. Zároveň tento způsob přináší riziko vzniku chyb z nedůsledného ošetřování vstupů SQL dotazu, což může vést až např. k SQL injection.

Kvůli zmíněným důvodům byla vytvořena třída *DB*, která zastřešuje základní SQL dotazy do sady metod s parametry. Parametry dotazů jsou zároveň pečlivě ošetřeny, aby neobsahovaly nebezpečné znaky. Třída obsahuje základní metody pro vkládání řádků do databáze, aktualizaci, mazání, výběr všech atd. Aby mohla být použita v praxi, byly v průběhu vývoje postupně přidávány obecné metody, například pro výběr specifikovaných řádků.

Při navrhování této třídy bylo dbáno na obecnost, aby mohla být v případě potřeby využita i v dalších projektech jako vrstva mezi nízkoúrovňovou databází a samotnou aplikací.

3.6 Obecná editace dat

Samotný Řídicí systém se skládá z modulů, které jsou v aplikaci nazývány *Handlers* (např. UsersHandler, TasksHandler, InfoServiceHandler atd), neboť *obsluhují* (angl. *handle*) dané datové entity. Pro odlišení od modulů jakožto autonomních aplikací (součástí generátoru), jsou označovány jako *obslužné moduly*.

V průběhu návrhu se ukázalo, že mnoho součástí (*obslužných modulů*) Řídicího systému (a tedy i mnoho datových entit) bude mít stejné nebo obdobné vlastnosti (atributy i metody). Například možnost přidávat nové záznamy bude mít jak modul pro správu uživatelů, tak modul pro správu procesů a stejně tak modul pro správu úloh. Kromě přidávání to bude i aktualizace dat, jejich výběr, výpis apod. Tyto obslužné moduly se budou lišit prakticky jenom atributy, které se budou různě jmenovat a budou různých typů.

Toto vše bylo vzato v úvahu a vnikla obecná třída, tedy jakýsi obecný obslužný modul, který všechny základní metody a jeden základní atribut (jednoznačné ID) již obsahuje. Ostatní moduly mohou potom z tohoto obecného dědit tyto základní vlastnosti a pouze je rozšiřovat o vlastní, specifické.

3.6.1 Modul pro správu uživatelů

K základnímu obslužnému modulu, ze kterého je tento modul zděděn, se definují navíc atributy, které bude každý uživatel potřebovat. Bude to login, heslo, jméno, příjmení, kontaktní informace (e-mail, telefon), příznak zda je aktivní a možnost přidělení administrátorských práv. Vedle těchto atributů je základní modul rozšířen o jednu metodu, která se stará o přihlášení uživatele, resp. pouze zjistí, zda login a heslo odpovídají existujícímu a aktivnímu uživateli.

3.6.2 Úlohy a podúlohy

Obslužný modul Tasks (úlohy) umožňuje správu úloh. Každá úloha obsahuje vstupní data a vždy patří právě jednomu uživateli. Při vytvoření, spuštění nebo zastavení úlohy se pro pozdější analýzu zaznamenává čas těchto akcí. Důležitý atribut je priorita úlohy, která posléze určuje, které podúlohy se budou provádět prioritněji.

U editace úlohy lze zároveň přidat možnost nechat se v určitých momentech informovat pomocí e-mailu nebo zprávy Jabber. Jedná se vlastně o propojení modulů Tasks a InfoService, které se kvůli jednodušší použitelnosti editují společně. Vždy lze nastavit, při jaké příležitosti a jakým způsobem se má uživatel informovat.

Záznamy dalšího modulu SubTasks (podúlohy) vždy patří právě jedné úloze, tedy jinak řečeno jedna úloha se skládá z mnoha podúloh, které jsou nově vytvářeny v průběhu vykonávání úlohy. Tento modul prakticky implementuje frontu podúloh, včetně výběru z fronty a její spuštění, kde pořadí podúloh rozhoduje, která podúloha bude z fronty vybrána jako první (vybírá se ale vždy jen z podúloh nejprioritnější úlohy).

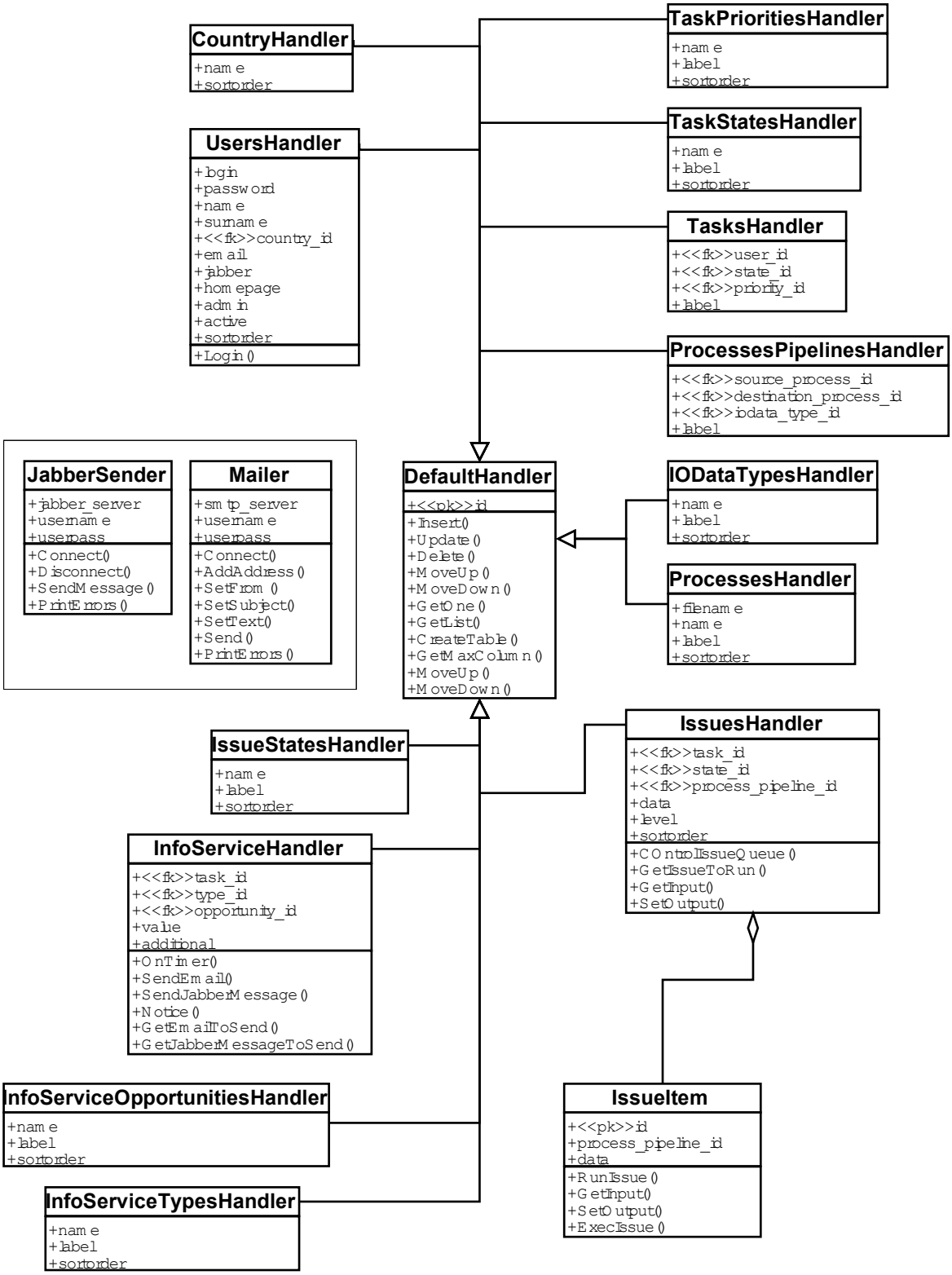
3.6.3 Procesy a cesty mezi nimi

Tyto obslužné moduly slouží k definování seznamu obslužných aplikací, které mohou zpracovávat podúlohy. Jsou zde definovány jejich atributy nutné ke spuštění aplikací, jejich vstupy a výstupy i pojmenování pro jednodušší identifikaci. Těmito aplikacemi mohou být např. *modul pro indexaci PDF*, *modul pro vyhledání publikací*, *modul pro nalezení homepage* autora atd.

Definování cest je nutné k jejich vzájemnému propojení, tedy například lze určit, že výstupní data modulu pro vyhledání publikací (tedy samotné názvy publikací) budou zároveň vstupem modulu pro indexaci PDF apod.

3.6.4 Ostatní obslužné moduly

Řídicí systém obsahuje kromě výše popsaných obslužných modulů ještě řadu jiných modulů, které ve většině slouží jako hodnoty SelectBoxů. Bylo by možné tyto moduly nahradit statickým seznamem



Obrázek 4: Diagram použitých modulů a některých jejich atributů a metod. V naprosté většině jsou moduly děděny ze základního modulu DefaultHandler.

hodnot, ale nebylo by potom možné tyto hodnoty editovat bez nutnosti zasahovat do kódu. Zvolené řešení je tedy obecné, kdy přidání nebo ubrání například priority úlohy lze provést pohodlně v administraci Řídicího systému.

Na obrázku č. 4 je diagram všech použitých obslužných modulů. Diagram ale neobsahuje všechny atributy a metody, slouží jen jako přehled.

3.6.5 Cizí klíče

Často se u definice sloupců setkáváme s tím, že se jedná o cizí klíč. Otázkou je, jak takovou hodnotu zobrazit. Vzhledem k tomu, že cizí klíč je použit například u příslušnosti jisté úlohy k uživateli, nebo podúlohy k úloze, nabízí se místo zobrazení konkrétního ID zobrazit SelectBox, kde budou na výběr právě uživatelé, resp. úlohy.

U takového vztahu je ale nutné definovat sloupec, který se bude v SelectBoxu zobrazovat jako popisek zobrazených prvků, zatímco o hodnotách jednotlivých prvků SelectBoxu není sporu, musí se jednat o jednoznačné ID.

3.6.6 Editace závislých položek

Aby bylo možné dobře vysvětlit tento problém, byly zvoleny dva konkrétní obslužné moduly, na kterých je vše lépe pochopitelné. Modul Tasks (úlohy) obsahuje základní vlastnosti úloh a modul InfoService obsahuje údaje nutné k informování uživatele o stavu prováděné úlohy (uvádí se příležitost informování a jeho způsob).

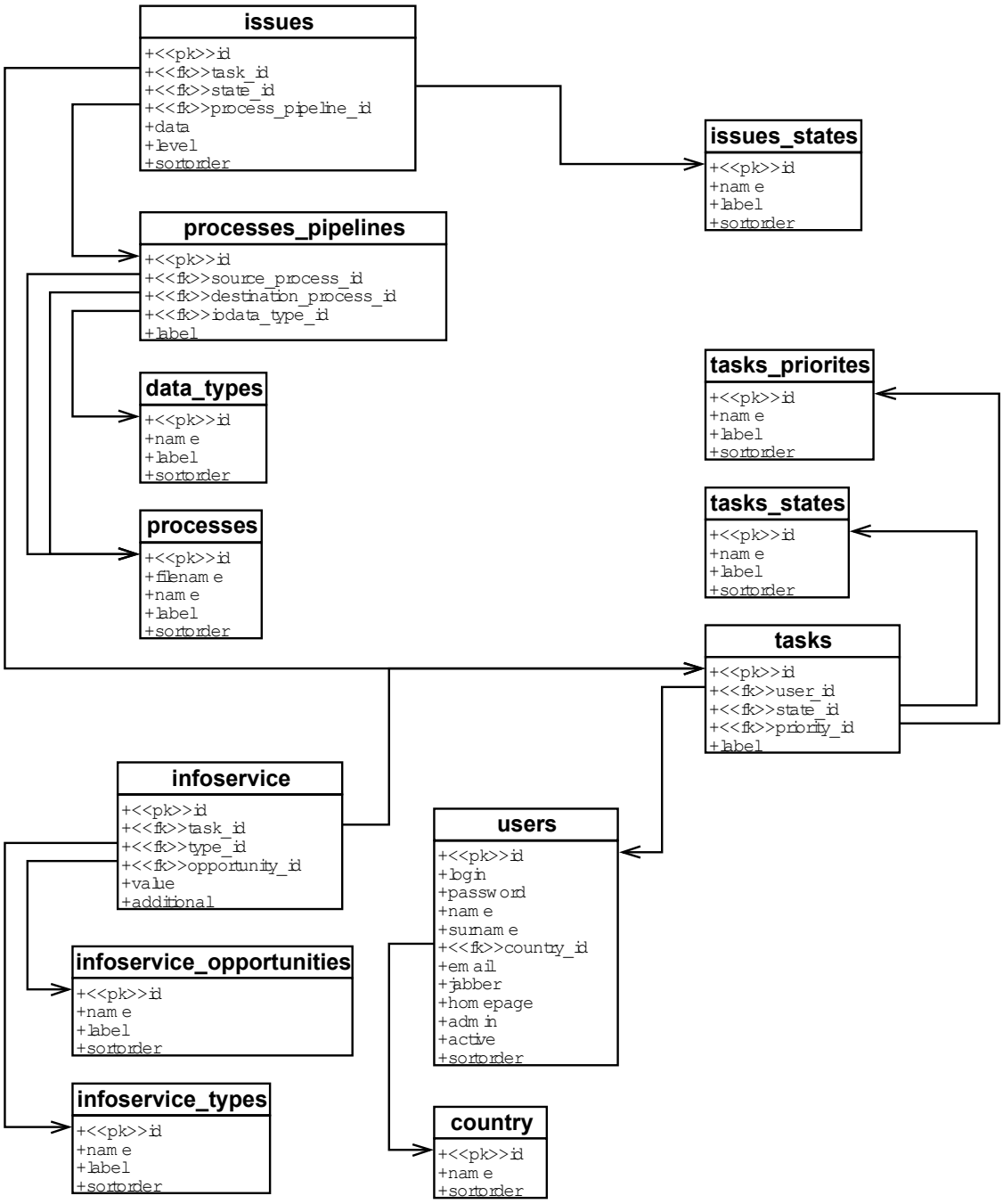
Tedy závislou položkou je myšlena taková položka modulu InfoService, která obsahuje cizí klíč do modulu Tasks, ale z důvodu pohodlnější editace se má zobrazovat právě v editaci položky modulu Tasks. Neboli v editaci nějakého záznamu modulu Tasks se budou přidávat, zobrazovat a editovat i záznamy tabulky InfoService.

Při editaci jisté položky modulu Tasks se musí tím pádem načíst i relevantní položky z modulu InfoService. Stejně tak při zpracování formuláře se položky InfoService musí zpět uložit do databáze.

Tímto zmíněným způsobem bylo docíleno velmi obecného řešení v oblasti zobrazení a zpracování formuláře, kde základem správného zobrazení všech dat je pouze správná definice atributů obslužného modulu.

3.6.7 ERD diagram databáze

Uložení dat v aplikaci znázorňuje diagram na obrázku č. 5. Zobrazeny jsou všechny použité tabulky se všemi sloupci a vztahy mezi tabulkami.



Obrázek 5: Diagram ERD vnitřní databáze Řídicího systému se znázorněním vztahů mezi tabulkami použitím cizích klíčů.

Z obrázku je zřejmé, že diagram do velké míry odpovídá struktuře na obrázku č. 4. Je tak dobře vidět, že každý obslužný modul má ve své podstatě v databázi svou tabulku, kam se ukládají právě data tohoto modulu.

3.6.8 Automatická inicializace databáze

Aby se mohlo s tabulkami vůbec pracovat, musí se nejprve vytvořit, a to samozřejmě spolu s definicí jejich sloupců. Původně byl v aplikaci obsažen konstruktor celé databáze. V případě, že se nějaká tabulka změnila, se musel tento konstruktor aktualizovat. Kromě aktualizace konstruktoru se ale musela i celá databáze vymazat a znovu vytvořit, aby se nový konstruktor do struktury promítl, a to nebylo velmi pohodlné.

Pro zefektivnění a zjednodušení práce byl proto zvolen jiný způsob. Struktura databáze se kontroluje při každé inicializaci démona, tedy při spouštění Řídicího systému. Nejdříve se kontroluje, zda tabulka existuje, a potom se kontrolují všechny sloupce, zda existují. Vzhledem k tomu, že typ sloupců má v SQLite3 prakticky doporučující charakter, nekontroluje se již typ sloupců.

Přidání sloupce nebo celé tabulky potom není v důsledku vůbec potřeba obstarávat a vše se děje automaticky. Pokud se chce celá databáze vymazat, není problém celý soubor s databází jednoduše odebrat. Soubor se totiž při dalším spuštění Řídicího systému opět automaticky vytvoří.

Obdobným způsobem jsou definovány základní data, která jsou pro běh systému nezbytná. Například by nebylo možné systém provozovat bez alespoň jednoho uživatelského účtu s administrátorskými právy. Při každé kontrole databáze je proto kontrolována i přítomnost alespoň jednoho takového účtu. Zbytek dat již lze definovat pomocí tohoto účtu.

3.6.9 Řazení prvků při výpise

Ve výpise například podúloh je jedním z důležitých atributů i pořadí podúlohy. Podle něho jsou totiž podúlohy vybírány, tedy nejdříve bude vykonána podúloha, která je zařazena jako první (má nejnižší pořadové číslo).

Aby se mohlo s pořadím podúloh pohodlně manipulovat, musí se i ve správném pořadí zobrazovat. Pokud je tedy nalezen atribut typu *sortorder*, budou se ve výpise položky řadit právě podle něho. Tím ale řazení prvků ve výpise nekončí, protože ne všechny tabulky (resp. obslužné moduly) obsahují atribut typu *sortorder*. Například modul *uživatelé* takový atribut nemají, přesto je potřeba je ve výpisu nějakým způsobem řadit, aby bylo snadnější najít konkrétního uživatele.

Pokud tedy tabulka neobsahuje přímo atribut s pořadím položek, lze nějaký atribut označit, že se má řadit právě podle něho. To lze udělat v konstruktoru konkrétního obslužného modulu, kde se definuje výčet atributů, jejich typ atd.

3.7 Zobrazování dat pomocí šablon

Webové uživatelské rozhraní používá pro zobrazování stránek modul publisher, který je volitelnou součástí modulu mod_python (viz. kapitola 2.1.8). Toto rozhraní podle zadané URL adresy zjistí, který skript a která metoda konkrétně se má na serveru začít vykonávat. HTML kód stránky, která se má zobrazit je potom návratová hodnota spuštěné metody. Základní otázkou tedy bylo, jak výsledný HTML kód generovat, aby byl co nejjednodušeji měnitelný a skript v pythonu byl co možná nejvíce čitelný.

První pokusy zjednodušit kód vedly k vytvoření metod pro generování hlavičky a patičky výsledné HMTL stánky. Toto řešení ovšem nestačilo, neboť generovat HTML kód přímo při načítání záznamů z databáze znamenalo vytvořit rozsáhlý kód, který už nebyl tak čitelný, aby toto řešení mohlo být bráno jako uspokojivé.

Vývoj celého webového rozhraní se tak začal ubírat jiným směrem. Malá inspirace byla nabrána z frameworků Ruby on Rails a Zope a šablonového systému Smarty. Vznikla nová koncepce, kde základním požadavkem bylo co možná nejvíce oddělit logickou část od prezentační.

Toto řešení se ukázalo jako velmi výhodné, neboť jak logická část, tak generování HTML kódu jsou samy o sobě relativně složité součásti a spojovat je dohromady by tuto složitost ještě umocnilo.

Obecně to funguje tak, že v logické části se vytvoří data, která se uloží do datové struktury. Tato datová struktura je předána modulu, který se stará o jejich zobrazení a jako výsledek vrací již výsledný HTML kód. Vzhledem k tomu, že s různými daty se z pohledu zobrazování nakládá různě, bylo vytvořeno několik základních verzí (šablon), jak uložená data zobrazit.

Vznikl tedy relativně jednoduchý šablonovací systém, který pouze generuje HTML kód zvolené stránky podle zvolené šablony a s předem definovanými daty.

4 Implementace Řídicího systému

V této kapitole budou popsány detaily samotné implementace, problémy, se kterými se autor v jejím průběhu setkal, a jejich řešení.

4.1 Úprava standardního XML-RPC serveru

Standardní knihovna XML-RPC nabízí téměř dostačující škálu prostředků pro práci s XML-RPC protokolem. Od vytvoření základního serveru a exportování metod na straně serveru, které mají být přístupné pomocí tohoto protokolu, až po připojení k takto vytvořenému serveru na straně klienta.

Problém ovšem nastal, pokud se takto rozběhlý server chtěl zastavit. Standardní server takovou možnost totiž vůbec nenabízí a muselo se tedy sáhnout k radikálnímu řešení – ukončit celý proces násilně.

To zajisté není ideální řešení, naštěstí existovala celkem jednoduchá náprava, která spočívala v rozšíření třídy XML-RPC serveru. Konkrétně se upravila metoda, kde server v nekonečném cyklu obsluhuje příchozí požadavky a byla implementována jedna nová veřejná metoda, která umožňuje ukončit XML-RPC server naprosto běžným a nenásilným způsobem.

4.2 Rozšíření výsledků formuláře

Při odesílání formuláře jsou odeslané hodnoty přístupny v základním objektu Request, který je exportován do většiny metod. Odeslaná data jsou tak přístupná téměř všude. Data jsou uložena ve slovníku, kde klíčem je název pole formuláře. Tento přístup velmi připomíná zpracování dat z formuláře například v PHP.

Při absenci nějakého klíče a pokusu o jeho přečtení ale vzniká vždy výjimka, která se může buďto obejít explicitní kontrolou na přítomnost zadaného klíče ve slovníku, nebo blokem try–except, který je v jazyce python velmi využívanou konstrukcí, často i v tomto případě. Existuje ale ještě jedna možnost, která dovolí pohodlně přistupovat k datům odeslaným formulářem, aniž by se musela předem nějak kontrolovat nebo ošetřovat blokem try–catch.

Byla vytvořena třída, která obaluje základní atribut typu slovník vlastními metodami na výběr prvku, a kde se provádí zmíněná kontrola na přítomnost hledaného klíče. Při použití této třídy se již kontrola provádět nemusí a v případě, že zadaný klíč neexistuje, je vrácen prázdný řetězec.

4.3 Vícenásobné prvky formuláře

Při zpracování formuláře se mohou objevit případy, kdy se jednotlivé názvy polí volí jako prvky pole (název elementu *input* jsou potom např. „*foo[1]*“, „*foo[4]*“ atd.). Tímto způsobem lze například v PHP jednoduše přistupovat k dynamicky generovaným elementům formuláře. Přistupuje se k nim, jako by se skutečně jednalo o pole (např. *$_POST['foo'][1]*, *$_POST['foo'][4]* atd.).

Ve zpracování podobných elementů formuláře v pythonu by se možná čekalo stejné chování, tedy že tyto prvky budou k dispozici jako *req.form['foo'][1]*, *req.form['foo'][4]* atd. Nicméně standardní modul mod_python tuto možnost neposkytuje a jméno elementu formuláře žádným způsobem neupravuje. Hodnoty formuláře jsou potom v základu k dispozici jako *req.form['foo[1]']*, *req.form['foo[4]']* atd.

Pro zpracování vícenásobných elementů stejného jména se ale daleko lépe hodí varianta první, takže nezbylo než tuto vlastnost doimplementovat. Před samotným zpracováním stránek se tak tyto hodnoty zpracovávají regulárními výrazy a to tak, že dokud se ve jméně prvku nachází hranaté závorky, jsou z těchto hodnot vytvářeny odpovídající slovníky. Tímto elegantním způsobem jsou potom hodnoty přístupné v obdobném stylu, jako je tomu např. právě v jazyku PHP.

4.4 Vícenásobná instance třídy Session

Autentizace na webovém uživatelském rozhraní je podmíněna přihlášením pomocí loginu a hesla. Informace o přihlášení je nutné uchovávat na stránkách jako kontext mezi jednotlivými stránkami. Proto se použití třídy Session, kterou modul mod_python v základu nabízí, ukázalo jako výhodné. V instanci této třídy je po dobu, kdy je uživatel přihlášen, uloženo jeho jednoznačné ID.

Práce s třídou Session se zdála být bezchybná až do chvíle, kdy skripty začaly bez zjevné příčiny zůstávat spuštěné. Opakovaným testováním a zároveň pročítáním diskusních fór [11] byl nakonec problém odhalen. Jednalo se o to, že instance třídy Session musí být vytvořena maximálně jedna v každém http požadavku. Pokud se při zpracování vytvořila druhá instance této třídy, skript zůstal nečinný, aniž by do prohlížeče odeslal jakákoliv data. Některé názory upozorňovaly na chybu v modulu, některé na problémy s uváznutím při zamykání přístupu k Session.

Nicméně ať je již tato tento stav způsoben čímkoliv, bylo nutné vytvářet instanci session pouze jednou za celou dobu zpracování jednoho požadavku. Proto je tato jediná instance vytvořena ještě před samotným spuštěním modulu publisher, v modulu *_startuphandler*.

4.5 Modul Startup handler

V průběhu implementace se ukázalo, že některé úkony je vhodné (někdy dokonce nutné) provést na každé stránce ještě před zpracováním a zobrazením dat.

Všechny tyto problémy vyřešil modul, který je spouštěn před samotným spuštěním modulu *publisher*. V tomto modulu se inicializuje i první a jediná instance třídy Session (o tomto problému se lze více dočíst v předcházející kapitole). Tato instance se uloží do proměnné request, která je předávána do každé metody zpracujícího skriptu. Přístup k ní je potom jednoduchý a není nutné vytváření nové instance pokaždé, když je nutné k sessions přistupovat.

4.6 Ukládání citlivých informací

Přesto, že veškerá data aplikace se ukládají do databáze, je relativně triviální se k těmto datům dostat, neboť databáze není chráněna. Uživatelské heslo je ale nutné nějakým způsobem chránit, neboť se jedná o velice citlivou informaci. I kdyby byla databáze šifrovaná, ukládání nešifrovaných hesel do databáze je obecně velmi nebezpečné – ať již z důvodu odcizení, tak z důvodu ochrany citlivých osobních informací. Stále mnoho uživatelů totiž k různým účtům používá totožná hesla.

Proto je nutné heslo šifrovat. V aplikaci je použitý jednostranný *hashovací* algoritmus md5. Heslo je zašifrováno zmíněným algoritmem a uloženo v 32-znakém řetězci. Při změně hesla (například pokud uživatel svoje zapomene) je tedy nutné zadat heslo nové, původní nelze již nijak zjistit ani obnovit.

4.7 Informační služba

Řídicí systém generátoru má mít implementovánu informační službu, která bude prostřednictvím různých protokolů zasílat informace o prováděných úlohách. V současné chvíli jsou implementovány způsoby dva: *zasílání e-mailu* a *zpráv pomocí protokolu Jabber*.

Možnost zasílat informace pomocí tohoto modulu je možné nastavit u každé úlohy. Nastavuje se důvod odesílání (např. při skončení úlohy) a způsob – e-mail nebo zpráva Jabber. V případě vybrání e-mailové zprávy je potřeba zadat navíc e-mailovou adresu, kam zprávy zasílat. V případě zvolení zpráv protokolu Jabber, je nutné zadat JID na serveru, na kterém má aplikace vytvořen účet (momentálně je to server Jabbim.cz).

Síťová komunikace a spolupráce s jinými servery, která je při zasílání e-mailů nebo zpráv Jabber nezbytná, nese nebezpečí ve formě pozastavení provádění aplikace až na několik sekund, což je velmi nežádoucí pro takovou aplikaci, jako je sám Řídicí systém generátoru. Oba moduly (odeslání

e-mailu a Jabber zprávy) se proto spouštějí v novém vláknu (procesu). Po spuštění těchto menších aplikací se tyto dotáží na vstup (komu, co a jak doručit) pomocí protokolu XML-RPC a jakmile zprávu odešlou, ukončí se.

4.7.1 Odesílání zpráv e-mailem

Pro odesílání e-mailů byla vytvořena obecná třída, která umí sestavit e-mail do správného tvaru a která využívá komunikace přímo s SMTP serverem [6], definovaným v konfiguračním souboru aplikace.

Při vytváření nového e-mailu byla lehká inspirace převzata z open-source projektu PHPMailer [12]. Pro testování byla vytvořena e-mailová adresa na veřejném freemailovém serveru Gmail [9]. Ten nabízí i vlasntí SMTP server, který je chráněn autentifikací. Ta je proto zahrnuta i do implementace této třídy.

4.7.2 Odeslání zpráv Jabber

Odeslání zprávy pomocí protokolu Jabber by pomocí standardních metod bylo oproti odeslání e-mailu značně složitější. Naštěstí se objevila možnost využít již vytvořenou open-source knihovnu jabber.py [16], která práci s Jabberem velmi zjednodušuje.

Pro testování byl vytvořen účet na veřejném, českém Jabber serveru Jabbim.cz. Pokud chce být uživatel informován pomocí tohoto protokolu, musí mít i on účet na tomto serveru.

4.8 Pravidelné události použitím timeru

Při inicializaci démona Řídicího systému je ve vlastním vláknu spuštěn timer, který po definované době (standardně 3s) provede určitou činnost. Činnosti lze přidávat do obslužné metody *OnTimer* (součást démona), přičemž metody se zpravidla volají přes protokol XML-RPC.

Momentálně je pomocí timeru spouštěna metoda, která kontroluje fronty podúloh všech uživatelů. Pokud nejsou prostředky zcela vyčerpány a ve frontě se nacházejí nějaké nezpracované podúlohy, začne se zpracovávat první v pořadí.

4.9 Definice zobrazování atributů

V průběhu implementace bylo nutné vyřešit zobrazování atributů v různých uživatelských rolích. Například atributy, které říkají, zda je uživatel aktivní nebo má administrátorská práva, smí editovat i vidět pouze administrátor, ale ne již uživatel ve svém profilu. Podobných atributů je v systému více, takže bylo nutné použít nějaký obecný přístup.

Problém byl vyřešen tak, že v definici atributů v konstruktoru obslužného modulu se kromě typu a jiných vlastností editují i práva, zda se má atribut zobrazovat u obyčejného uživatele nebo jenom u administrátora, případně vůbec (nikdy se např. nezobrazuje ID).

S tím souvisí i další problém se zobrazováním atributů, tentokrát ve výpise seznamu prvků. Často mají totiž tabulky (resp. obslužné moduly) mnoho atributů a zobrazování naprosto všech ve výpise je nežádoucí, neboť by výsledná tabulka byla zbytečně rozsáhlá a tím pádem nepřehledná.

I tento problém byl vyřešen obdobně, jako předchozí – tedy v konstruktoru obslužného modulu se definuje i viditelnost atributu ve výpise prvků.

4.10 Ošetření dat z databáze

Vzhledem k tomu, že při vkládání do databáze jsou data ošetřena pouze proti SQL injection, může být v databázi uloženo mnoho dat, která by ve výpisu vedla ke zhroucení stránky (např. šikovný javascriptový kód, část HTML kódu apod.). Data jsou proto před výpisem ošetřena tak, že jsou z nebezpečných znaků vytvořeny tzv. HTML entity [10], které tento problém odstraní.

4.11 Zadávání nového hesla

Při obecné definici atributů obslužného modulu lze nějakému atributu nastavit příznak, že při jeho editaci je nutná kontrola. Systém se automaticky postará o to, že se daná položka zobrazí dvakrát, a data nebudou označena jako validní, pokud hodnoty v obou položkách nebudou shodné.

Tímto způsobem lze elegantně definovat položku typu *heslo*, kde je při vložení nebo změně položky nutné heslo znovu zopakovat, aby se minimalizovala možnost překlepu.

5 Závěr

Tvorba tak rozsáhlého systému, jako je Řídicí systém generátoru vědeckých webových portálů, si zasloužila velkého úsilí a maximální pozornost zejména při analýze, neboť špatná nebo neobecná analýza mohla celý projekt velmi ztížit.

Při implementaci se ukázalo, že první pokusy bez rozsáhlejšího návrhu neměly dlouhého trvání, a potvrdilo se zde i to, že důkladná příprava a často složitější, ale obecnější návrh řešení v důsledku ušetří mnoho práce.

Základní cíle této práce byly splněny, bohužel se nepovedlo celý projekt spojit dohromady nebo ho dokonce připravit na použití v praxi. Některé moduly totiž zatím nejsou implementovány a bez všech základních nemá smysl celý generátor provozovat. Nicméně díky použití co možná nejobecnějších metod návrhu a implementace nebude v budoucnu problém tyto části přidat bez výraznějších změn v samotném systému.

Vzhledem k tomu, že Řídicí systém je pravděpodobně nejdůležitější částí celého Generátoru vědeckých webových portálů, mělo by se mu i v budoucnu dostat nejvýraznější pozornosti i co se týká rozšíření. Pozornost by si zasluhoval propracovanější systém logování, zajisté by se mohly přidat další možnosti, jak interaktivně komunikovat s uživatelem. Kromě e-mailových zpráv a zpráv na Jabber by to mohly být zprávy na ICQ nebo jiné protokoly Instant messagingu. Případně by se mohla implementovat i podpora zasílání krátkých textových zpráv SMS na mobilní telefon nebo zasílání Jabber zpráv na libovolný veřejný server (kde by se musel vyřešit problém s automatickou registrací).

Pokud by se mělo vyvíjet webové uživatelské rozhraní, mohla by se přidat možnost řadit prvky podle libovolného sloupce, volit si počet zobrazených prvků ve výpise, přidat možnost hromadné editace záznamů atd.

V případě, že by se služba v budoucnu využívala v masovém měřítku stovkami nebo i více uživatelů, bylo by pravděpodobně nutné rozšířit službu na více výkonných serverů, kdy by se tím pádem musela řešit jejich vzájemná komunikace.

Literatura

[1] Harms, D., McDonald, K. *Začínáme programovat v jazyce Python*. Brno, ComputerPress, 2003.

[2] Lutz, M., Ascher, D. *Naučte se Python.* Praha, Grada, 2003.

[3] SQLite Consortium, *SQLite Documantation*. [online] [2008-04-29] Dostupné na <*http://www.sqlite.org/docs.html*>

[4] GeoWikiCZ, *Python - načtení a zpracování XML ze zadaného URL*. [online] [2008-04-29] Dostupné na <http://gama.fsv.cvut.cz/wiki/index.php?title=Python_-_na%C4%8Dten%C3%AD_a_zpracov%C3%A1n%C3%AD_XML_ze_zadan%C3%A9ho_URL&oldid=9283>

[5] Petrie, G., Petrie, P., Petrie, D. *Python and XML: An Introduction*. [online] [2008-04-29] Dostupné na <http://www.boddie.org.uk/python/XML_intro.html>

[6] Sheila *Sending email in Python*. [online] [2008-04-29] Dostupné na <http://www.thinkspot.net/sheila/article.php?story=20040822174141155>

[7] Jabber.org Team. *Jabber Resources*. [online] [2008-04-29] Dostupné na <http://wiki.jabber.org/index.php/Jabber_Resources>

[8] Python Library Reference. *Process Management*. [online] [2008-04-29] Dostupné na <http://docs.python.org/lib/os-process.html>

[9] Python mailing list. *Smtplib starttls gmail example*. [online] [2008-04-29] Dostupné na <http://mail.python.org/pipermail/python-list/2007-January/423569.html>

[10] Python mailing list. *Convert from unicode chars to HTML entities*. [online] [2008-04-29] Dostupné na <http://mail.python.org/pipermail/python-list/2007-January/424255.html>

[11] Mod_Python mailing list. *mod_python session trouble*. [online] [2008-04-29] Dostupné na <http://www.modpython.org/pipermail/mod_python/2005-May/018035.html>

[12] Matzelle, B. *PHP Mailer*. [online] [2008-04-29] Dostupné na <http://phpmailer.codeworxtech.com/>

[13] Nettleton, N. *Trim a string in javascript*. [online] [2008-04-29] Dostupné na <http://www.nicknettleton.com/zine/javascript/trim-a-string-in-javascript>

[14] OsDir. *Issues with "Session use with classes" wiki example*. [online] [2008-04-29] Dostupné na <http://osdir.com/ml/apache.mod-python.devel/2006-12/msg00038.html>

[15] SQLite Consortium, *Speed Comparison.* [online] [2008-04-29] Dostupné na <http://www.sqlite.org/cvstrac/wiki?p=SpeedComparison>

[16] Allum, M. *Python Jabber Library.* [online] [2008-04-29] Dostupné na <http://jabberpy.sourceforge.net/>

Seznam příloh

Příloha 1. Instalace Řídicího systému.

Příloha 2. Manuál k webovému rozhraní.

Příloha 3. CD se zdrojovými texty s tímto obsahem:

- Adresář */demon* – zdrojové texty pro spuštění démona Řídicího systému.
- Adresář */htdocs* – zdrojové texty webového rozhraní, určené pro nahrání do složky projektu na webovém serveru
- PDF soubor *technicka_zprava.pdf* – úplné změní technické zprávy ve formátu PDF

Příloha 1. Instalace Řídicího systému

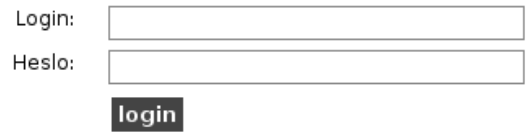
Instalaci démona Řídicího systému generátoru vědeckých webových portálů provedeme nakopírováním souborů ze složky */demon* uložených na CD (příloha 3.) do adresáře na serveru. V souboru */demon/Config.py* můžeme změnit přednastavené hodnoty nastavení serveru a portu, na kterém poběží služba protokolu XML-RPC. Zde je možné též změnit detaily SMTP a Jabber účtu pro zasílání informací o provádění úloh uživateli.

Instalaci webového uživatelského rozhraní provedeme obdobným způsobem, tedy nakopírováním souborů ze složky */htdocs* uložených na CD (příloha 3.) do adresáře na webovém serveru. V souboru */htdocs/Core/Config.py* potom změníme hodnoty *PROJECT_PATH* na absolutní cestu k souborům webového rozhraní a *PROJECT_BASE_URL* na url, na které bude uživatelské rozhraní k dispozici veřejně. Nastavení údajů pro přístup k XML-RPC serveru bude obdobné, jako v předcházejícím odstavci.

Příloha 2. Manuál k webovému rozhraní

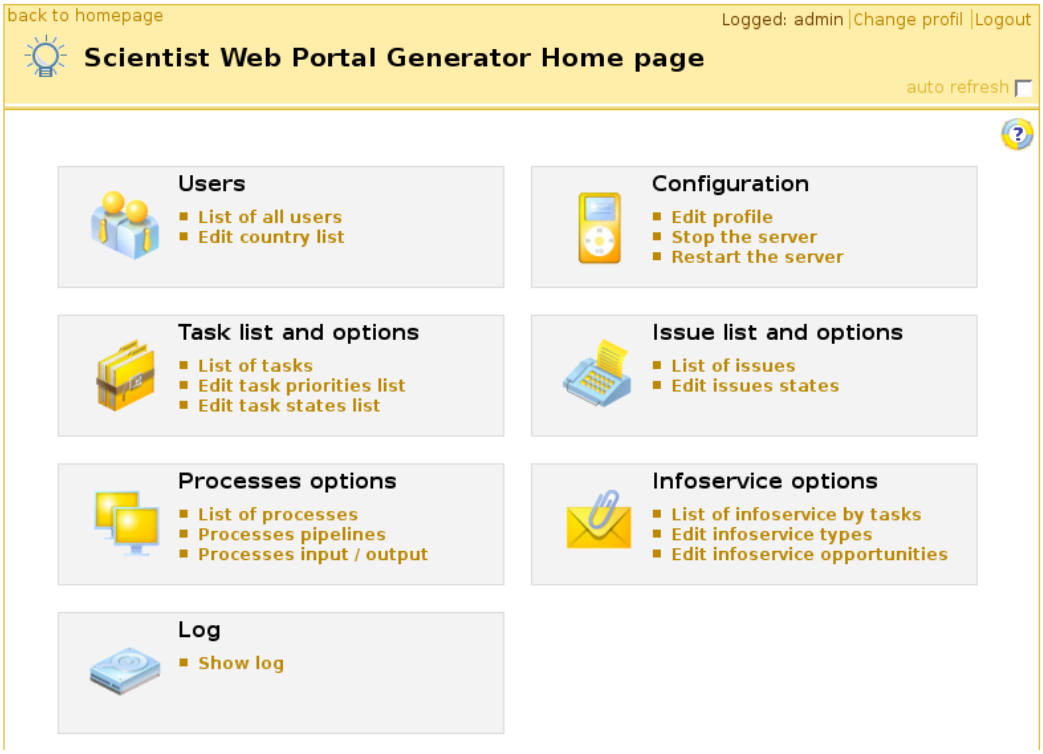
Login – Přihlášení

Okno s přihlášením se objeví pokaždé, pokud uživatel načte jakoukoliv stránku, ale není přihlášen. Po zadání správných přihlašovacích údajů a kliknutí na tlačítko „login“ bude uživatel přihlášen.



Main page – Hlavní rozcestník

Na hlavní stránku s rozcestníkem se uživatel dostane vždy kliknutím na logo v levé horní části stránky. Z hlavního rozcestníku se lze dostat na ostatní stránky aplikace. Seznam možných stránek se liší od uživatelské role, kdy administrátor má zpřístupněny všechny položky, zatímco běžný uživatel pouze některé z nich.



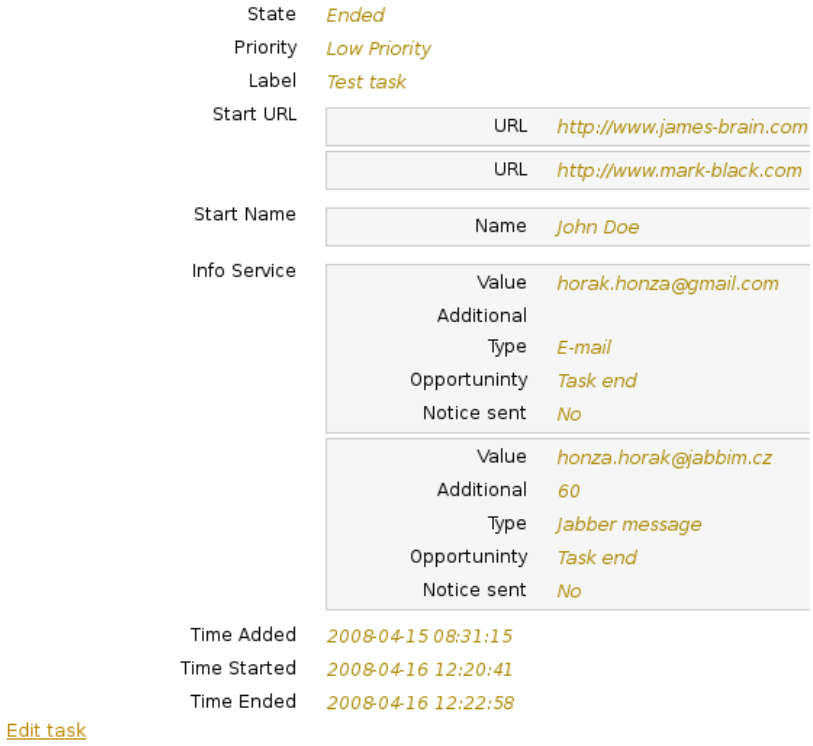
Users list – správa uživatelských účtů

Zde lze přidávat, editovat a mazat uživatelské účty. Mějme na mysli, že uživatelský účet musí být pro správné fungování aktivován. Při vytváření nového účtu se musí vyplnit login a heslo, které lze spolu s ostatními údaji následně změnit v profilu.

Běžný uživatel potom smí editovat pouze vlastní profil, zatímco administrátor smí editovat účty všechny.

Tasks list – správa úloh

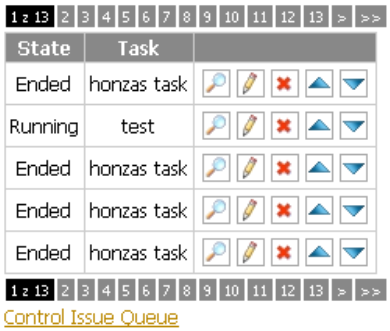
Zde je možné kontrolovat seznam úloh konkrétního uživatele i se základními atributy nebo vytvořit novou úlohu s definovanými vstupními daty. V detailu úlohy je pak možné danou úlohu spustit, případně pozastavit nebo předčasně ukončit.



Subtasks list – správa podúloh

Zde je možné vidět všechny podúlohy, které čekají ve frontě, jsou právě zpracovávány nebo jsou již zpracovány. Jejich pořadí lze měnit pomocí šipek v pravém sloupci.

Administrátor vidí všechny podúlohy, zatímco běžný uživatel pouze podúlohy přiřazené k vlastním úlohám.



Processes list – editace aplikací (modulů)

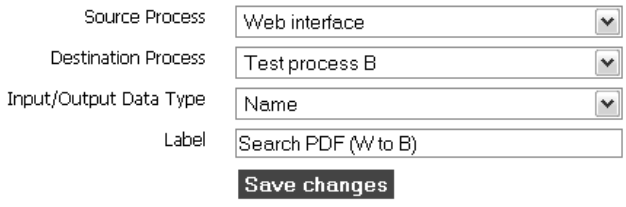
Zde se mohou editovat nebo přidávat nové moduly (aplikace) do generátoru vědeckých webových portálů. Důležité je správně definovat umístění spouštěcího souboru.

In/Out Data list – definice možných dat

Zde se definují data, která mohou mezi dvěma různými aplikacemi chodit. Zvolené jméno musí být ve tvaru, který dovoluje XML jako název elementu, protože právě podle tohoto atributu a jména elementu ve výsledném XML (výstupní data určité podúlohy) se zjišťuje, který modul bude vybrán pro další zpracování dat.

Processes Pipelines list – editace spojení aplikací (modulů)

Zde se editují spojení mezi moduly (aplikacemi), tedy které výstupy té které aplikace jsou navázány jako vstup jiné aplikace. Zvolí se tedy začáteční (start process) a konečný (end proces) modul a definují se data, jaká daným spojením mohou probíhat.



Log

Systém automaticky loguje některé úkony a pod touto položkou je možné prohlédnout si záznamy tohoto logu. Log lze využít i ke zpětnému dohledání provedených úkonů.

InfoService options

Zde je možné editovat nebo přidávat typy informačních služeb, tedy možnosti, kdy které oznámení zasílat atd. Změny v tomto modulu musí být zároveň doplněny změnami v kódu, kde je nutné do místa, kde se má o čemkoliv informovat, vložit vyvolání příslušné metody.

Rychlá nápověda a automatický refresh

V horním pravém rohu každé stránky nalezneme dvě malé pomůcky, které mohou zpříjemnit práci. První z nich je *rychlá nápověda*, která se po najetí na ikonku zobrazuje, případně lze po kliknutí na ikonku přejít přímo na stránku s nápovědou.

Druhou pomůckou je zaškrtávací políčko *auto-refreshe* stránky. Pokud na jakékoliv stránce políčko zaškrtneme, stránka se bude v pravidelných intervalech (standardně 3s) obnovovat. Použití se nabízí například pokud chceme sledovat průběh vytváření a provádění podúloh.